UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Estudio y diseño de un proceso de monitorización aplicado a un motor eléctrico por medio de la inteligencia artificial

Trabajo de Fin de grado
Grado en ingeniería de los Vehículos Aeroespaciales

Autora:
Barradas Barradas, Cristina

Director:
Delgado Prieto, Miguel

Codirector:
Arellano Espitia, Francisco

Convocatoria:

Enero 2020

Resumen

Actualmente nos encontramos en lo que muchos han llegado a denominar como la era de la tecnología. No en vano, las últimas décadas han estado marcadas indudablemente por los avances tecnológicos. En pocos años se ha pasado de necesitar a 4 hombres para poder mover un disco duro de tan solo 5Mb (¡lo equivalente a 6 imágenes!) a ordenadores de alta capacidad computacional que ni siquiera llegan a pesar kilo y medio. Pero lo que hace que todos estos avances tecnológicos sean tan importantes es sin duda su capacidad para cambiar completamente nuestro estilo de vida de forma permanente. Y es que, sin darnos cuenta nos hemos ido adaptando a todos estos avances y a día de hoy es complicado imaginar un mundo sin ellos.

Figura 1: Disco duro de 1956 [1]

Pero cabe preguntarnos, ¿Qué es lo que ha permitido este avance tan brutal y rápido? Pues bien, una de las áreas científicas artífice de esta incesante metamorfosis a la que está sometida la sociedad actual es, sin duda, la Inteligencia Artificial. El ámbito de aplicación de la inteligencia artificial es extremadamente amplio. En la industria del ocio es utilizado para analizar los motores de búsqueda realizados en cualquier plataforma, para la generación automática de recomendaciones personalizadas, o para crear rivales virtuales que llegan a superar al propio ser humano en gran cantidad de videojuegos. Pero gran parte de la importancia del aprendizaje autónomo reside sin duda en su capacidad para facilitarnos la vida. De esta forma, la inteligencia artificial se ha utilizado para la detección de fraudes en entidades como bancos, o para la detección de tumores en el campo de la medicina. Estos son solo unos cuantos ejemplos, pero la lista sería interminable.

Dentro del aprendizaje automático la técnica con mayor potencial de desarrollo en la actualidad sería el aprendizaje profundo o deep learning. Éste se basa en crear una estructura con la mayor similitud posible al del cerebro humano, de forma que el proceso de aprendizaje ocurra de forma análoga.

El propósito del presente trabajo es explorar el diseño, implementación y análisis de técnicas deep learning para la caracterización y reconocimiento de patrones. La máquina realizará esta caraterización de forma autónoma para su posterior clasificación, con el fin de implementar un sistema de autodiagnóstico y detección de fallos en el sistema.

En los primeros capítulos se realizará un proceso de investigación sobre las diferentes alternativas que se ofrecen. Posteriormente se dispondrá a realizar una correcta preparación de los datos de partida, se diseñará e implementará la red neuronal, y finalmente se realizará un análisis de los resultados obtenidos.

Índice

1. Introducción

1.1. Objeto del trabajo

El objetivo del trabajo es el de estudiar la implementación de técnicas de inteligencia artificial para el mantenimiento predictivo de un actuador electromecánico. Dicho actuador emulará el mecanismo de extracción y retracción de un elemento hiper-sustentador del ala de un avión. Este sistema podrá presentar durante su vida de servicio una serie de posibles estados de fallo:

- Demagnetización
- Desgaste de los cojinetes
- Excentricidad incorrecta
- Desgaste de los engranajes

Se programará una inteligencia artificial en deep learning capaz de realizar la caracterización de de los mencionados estados así como de su estado correcto de funcionamiento. El objetivo será conseguir una representación gráfica con la forma cualitativa de la Fig.2, donde se muestra un ejemplo de proyección de condiciones en dos dimensiones. Cada eje representará una característica numérica, como podrían ser la velocidad angular o los valores máximos posibles.

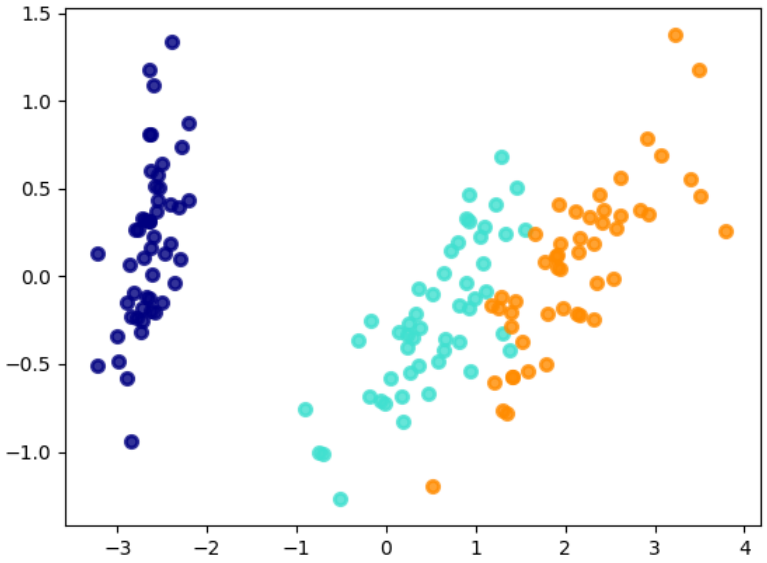


Figura 2: Caracterización de datos [2].

Una vez conseguido dicha caracterización según su condición de fallo, se hará una clasificación de los mismos con el fin de que el actuador sea capaz de autodiagnosticarse, es decir, identificar claramente cual es el problema que se está produciendo en un cierto momento. Por último mediante un correcto estudio del error producido al reconstruir la información inicial introducida, se tratará de comprobar si estas técnicas permiten tener un porcentaje de error lo suficientemente pequeño como para permitir su uso de forma natural, permitiendo una mayor rapidez en el diagnóstico de la pieza.

De forma más detallada, los objetivos del presente proyecto son:

- Investigación sobre las diferentes técnicas de inteligencia artificial en la actualidad.
- Preparación de la información a procesar.
- Diseño del modelo.

- Implementación del modelo.
- Evaluación de los resultados.
- Propuestas de mejoras.
- Obtención de conclusiones.

1.2. Alcance del trabajo

El alcance del trabajo será el diseño e implementación de un modelo en deep learning que se ajuste al problema que se quiere resolver (Sección 1.1). Para dicha implementación se utilizará el toolbox de deep learning del software computacional Matlab. Los datos corresponderán a los diferentes estados de fallo que podrá adoptar un sistema electromecánico. Estos datos corresponderán a una señal de vibración que harán las veces tanto de datos de entrenamiento como de validación y prueba.

Se realizará una reducción de dimensión de los datos de entrada con el fin de poder realizar una representación en dos dimensiones de los mismos. También se hará un estudio paramétrico para ajustar las diferentes variables con el objetivo de conseguir una salida del autoencoder con un error suficientemente pequeño.

Del mismo modo, se realizará un proceso de iteración para ajustar el número de capas ocultas en base a la minimización del error. Una vez realizada la representación de los datos se procederá a clasificar los mismos con el fin de que el sistema pueda autodiagnosticarse e identificar de forma autónoma que error se está produciendo. Finalmente se realizará un análisis de los resultados mediante la construcción de la matriz de confusión.

Los puntos principales de este proyecto serán:

- Introducción teórica
- Estudio paramétrico del autoencoder.
- Disminución dimensional de los datos de entrada.
- Estudio del error cuadrático medio para seleccionar el número de capas ocultas óptimo.
- Clasificación de los estados de fallo.
- Análisis de los resultados

1.3. Requerimientos del trabajo

Lo primero que será necesario disponer para la realización de este proyecto será de un conjunto de datos lo suficientemente grande como para que el algoritmo pueda aprender el comportamiento del sistema. De lo contrario, no se podrá realizar una reconstrucción con un error suficientemente pequeño. En este proyecto los datos serán proporcionados por los tutores de este trabajo.

Además también será necesario disponer del toolbox deep learning del software de programación Matlab, disponible a partir de la versión de matlab 2015 y que puede adquirirse directamente a partir de la página oficial de dicho software[3]. No obstante, lo desarrollado en este trabajo sería fácilmente extrapolable a otros lenguajes de programación.

1.4. Utilidad del trabajo

El primer problema que abordará este trabajo será el diseño general, implementación y análisis de técnicas deep learning para la caracterización y reconocimiento de patrones con el fin de conseguir un ahorro significativo de recursos en el mantenimiento de actuadores como el de objeto de estudio del presente proyecto. Más concretamente, el algoritmo que se desarrollará servirá para la temprana detección de fallos en un actuador electromecánico que emulará la actuación de un elemento hiper-sustentador del ala de un avión. A su vez, se

intentará que el modelo sea suficientemente general y con resultados aceptables, con el fin de que pueda ser implementado para diversas tareas reales.

En la actualidad los modelos de aprendizaje no supervisado se están especializando en el reconocimiento de imágenes y texto o sistemas de recomendación. No obstante en este proyecto se trabajará con vectores de datos numéricos que representarán señales periódicas que aportarán información sobre posibles fallos en un actuador electromecánico. El fin del análisis consistirá en que al final del proyecto, el sistema pueda identificar con un margen de error suficientemente pequeño, cual es el fallo que se está dando en un determinado instante.

Esta capacidad de autodiagnóstico es de enorme utilidad en sistemas de gran complejidad formados por la integración de diversos subsistemas menores. El autodiagnóstico se ha popularizado en actividades como la certificación de productos en la industria aeronáutica. Esto es consecuencia de la enorme reducción del tiempo de mantenimiento que traen consigo. Aunque es cierto que este tipo de técnicas requiere un alto gasto computacional inicial, una vez implementados facilitan enormemente las tareas de detección y en ocasiones incluso de reparación. Como los fallos son detectados mediante un software al que se le introducen una serie de datos extraídos directamente del sistema, el proceso de desmontaje de piezas es reducido al mínimo. Así mismo, se eliminan por tanto todas las complicaciones derivadas como podría ser la rotura de piezas con uniones no desmontables, holguras y tolerancias pequeñas y difíciles de conseguir en el posterior montaje o la mera seguridad de los operarios.

2. Estado del arte

2.1. Antecedentes

Es complicado hablar del estado del arte de la inteligencia artificial, ya que no existe un algoritmo común capaz de encontrar la solución para cada tipo de problema. Las necesidades de dichos algoritmos varían considerablemente según los requerimientos propios del proyecto.

Si bien, es cierto que durante los últimos años han aparecido multitud de técnicas que aportan diversos tipos de posibles soluciones. Estos tratan de solucionar todo tipo de problemas complejos y en general aquellos que requieren de un gran procesamiento de información por ser función de un incontable número de variables.

2.1.1. Historia

El concepto de inteligencia artificial hace referencia al aprendizaje de un ordenador a partir de la experiencia, sin depender de ecuaciones o modelos matemáticos predefinidos. Esto tiene su base en el modelo de aprendizaje natural de los seres humanos, capaces de procesar una gran cantidad de información con el fin de encontrar patrones que les permitan predecir situaciones y tomar mejores decisiones.

Desde el principio de los tiempos el ser humano ha sentido fascinación por el funcionamiento del cerebro humano. Como puede verse en las múltiples obras de los filósofos clásicos, ya en la antigua Grecia Platón (427-347 a.C) reflexionaba sobre la mecánica del pensamiento humano. Sin embargo no fue hasta 1936 cuando, el que es considerado uno de los padres de la ciencia de la computación y precursor de la informática moderna, Alan Turing, estudió el cerebro como una forma de entender el mundo de la computación.[4]

A partir de entonces numerosos científicos han perseguido el sueño de crear máquinas con capacidad de aprendizaje inteligente que pudieran realizar todo tipo de tareas de forma autónoma. Aunque este concepto haya podido ser erróneamente malinterpretado en numerosas ocasiones en la gran pantalla con la idea de autómatas que podrían llegar a suponer una amenaza real en un futuro no tan lejano, al más puro estilo de Terminator 3. [5]. No obstante, la verdadera funcionalidad de las máquinas controladas por inteligencia artificial es que sean capaces de ayudar al ser humano en todo tipo de tareas. Especialmente en aquellas que puedan entrañar cualquier tipo de riesgo para el operario o tareas excesivamente repetitivas y tediosas.

En la actualidad este tipo de técnicas están tan presentes en nuestra vida cotidiana que con frecuencia ni siquiera somos conscientes de su gran utilidad. Sus aplicaciones son muy diversas. Desde el reconocimiento facial, de voz o incluso de huella dactilar implementada en cualquiera de nuestros teléfonos inteligentes o hasta su papel fundamental en campos como la medicina, donde la inteligencia artificial ha sido utilizada para la detección de todo tipo de enfermedades.

2.1.2. Neuronas Artificiales

Las redes neuronales artificiales intentan reproducir los componentes básicos de una red neuronal biológica. En el cerebro la unidad estructural básica podría decirse que es la neurona, y del mismo modo, en la inteligencia artificial, estas neuronas serán creadas mediante programación computacional. Estas neuronas artificiales se interconectarán con el fin de crear redes neuronales que transmitan los datos de una a otra. [6]

Siguiendo el funcionamiento de un sistema neuronal biológico, una neurona artificial se caracterizará por:

- Entrada: constituirá los datos recibidos desde la salida de otra neurona previa.
- Conexión: cada entrada será fruto de una conexión que se caracterizará por una serie de parámetros que le otorgarán mayor o menor peso.
- Valor umbral: Una vez todas las entradas llegan a la neurona, cada una es multiplicada por su peso y posteriormente serán sumadas. Si esta operación sobrepasa un cierto valor umbral se producirá la activación de la neurona.
- Función de transferencia: procesa la información recibida y genera la salida de la neurona.

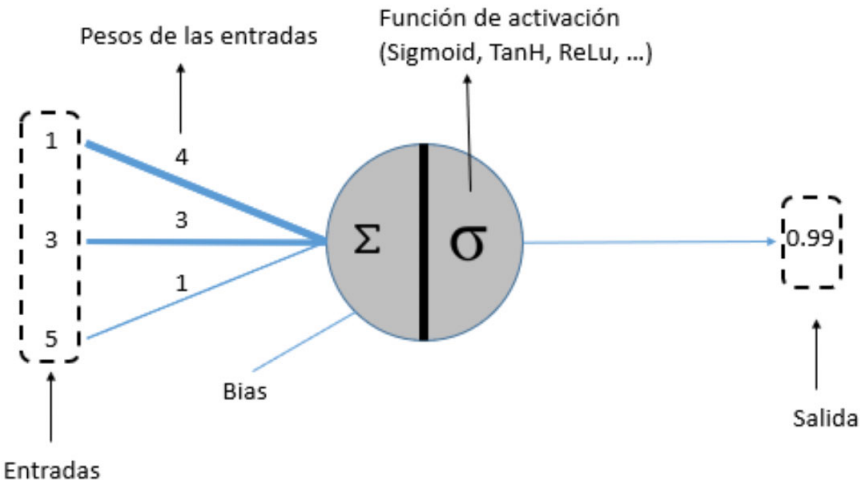


Figura 3: Neurona Artificial [7].

2.1.3. Redes neuronales artificiales

Sin embargo por sí mismas las neuronas no tienen la capacidad de resolver problemas complejos como sí lo hacen las redes de neuronas artificiales. Es por ello que las neuronas forman miles y millones de interconexiones entre ellas.

Aunque las redes neuronales artificiales están basadas en la estructura de un cerebro biológico, debido a su gran complejidad y la falta de exactitud en su conocimiento, replicar un cerebro humano es hasta la fecha, algo imposible. No obstante se han conseguido excelentes resultados por parte de las redes neuronales artificiales en tareas de reconocimiento y agrupación de todo tipo de conjuntos masivos de información. Es obvio que una red neuronal no puede pensar como un ser humano ya que no disponen de pensamiento creativo pero como ya se ha demostrado, sí que pueden aprender y ser entrenadas.

En cuanto a la estructura, una red neuronal artificial está formada por múltiples capas de procesamiento interconectadas. Estas capas están formadas por multitud de sistemas no lineales (integrados por cientos de neuronas) que procesan la información de forma independiente. En la Fig.4 pueden verse las diferentes partes de dicha red.

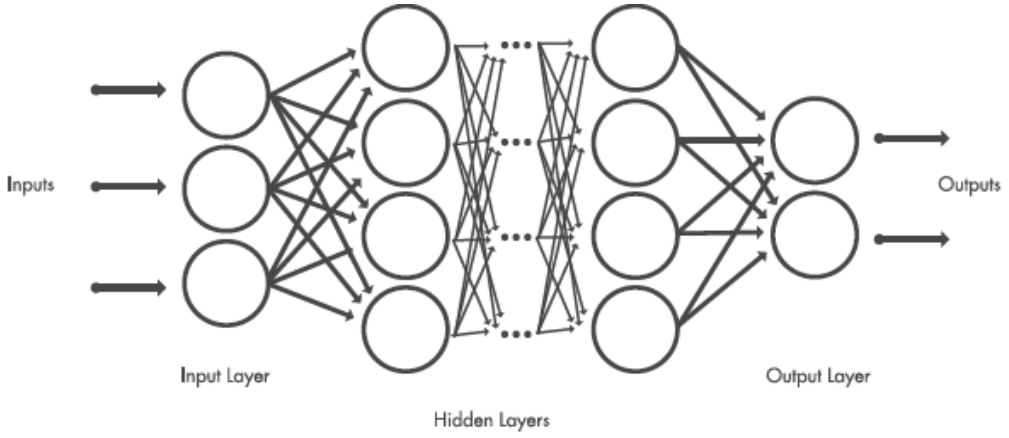


Figura 4: Estructura de la red neuronal [3].

Las primeras capas de entrada no tienen propiedades neuronales, ya que únicamente reciben los datos reales de partida mientras que la capa de salida proporcionará los resultados visibles de la red. Las capas intermedias entre la entrada y la salida son denominadas capas ocultas, ya que serán el "*cerebro*" de la red neuronal artificial, la cual aprenderá el proceso a desempeñar tras su correcto entrenamiento, pero cuyo funcionamiento interno será en todo momento desconocido. El proceso de aprendizaje consistirá en un proceso iterativo por el cual se irán ajustando los pesos y el umbral de la red con el fin de mejorar la resolución de los resultados deseados.

2.2. Planteamiento y decisión sobre soluciones alternativas

Dentro de los diversos métodos de inteligencia artificial, existen múltiples tipos de técnicas y algoritmos. Por ello la tarea de decidir cual de ellas es la más adecuada para un cierto problema dado puede parecer un problema ciertamente complicado de abordar inicialmente. No obstante, no existe un método que sea "el mejor". Para determinar el algoritmo óptimo no existe otra forma que comprobarlo experimentalmente. Pero además, la selección del algoritmo también dependerá del tamaño y tipo de datos con los que se esté trabajando así como de la información que se desee obtener.

Por todo esto, cualquier trabajo de machine learning debe comenzar con tres preguntas:

- ¿Con qué tipo de información se está trabajando?
- ¿Qué tipo de información se quiere conseguir?
- ¿Cómo se utilizará la información obtenida?

Respondiendo a estas tres preguntas podremos decidir qué tipo de aprendizaje debemos utilizar de entre las diferentes clasificaciones que podemos ver en la Fig.5

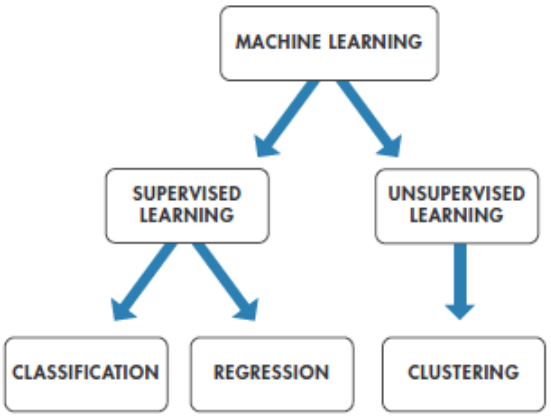


Figura 5: Clasificación técnicas de Machine Learning [8].

2.2.1. Supervised Learning

Un algoritmo de aprendizaje supervisado toma un cierto conjunto de datos de entrada conocidos y otro conjunto de respuestas del mismo modo conocidas que constituirán las salidas del sistema. Con toda esta información se entrena el modelo para generar predicciones con el menor error posible a nuevos datos de entrada. Todas las técnicas de learning machine supervisado son utilizadas para la clasificación de información o para la predicción de una sucesión continua de información (regresión).

Este tipo de aprendizaje es útil cuando se dispone de suficiente información sobre la salida que la máquina intentará replicar. Este tipo de técnicas también es conveniente cuando se tratan datos con poca variabilidad o viene dada en cantidad moderada.

Figura 6: Técnicas supervisadas de ML para regresión y clasificación [9].

Un ejemplo de este tipo de técnicas podría ser el introducir una gran cantidad de imágenes de gatos a la máquina, bajo la consigna "gato". Así la red neuronal comenzará a memorizar qué imágenes se corresponden con la representación de un gato. Cuando posteriormente se le muestre una imagen, deberá dictaminar si se le está mostrando un gato o no.

2.2.2. Unsupervised Learning

El aprendizaje no supervisado es útil cuando se quiere estudiar un conjunto de datos pero no se sabe bien el propósito específico de su uso o no se conoce con exactitud la información que contienen los datos de partida.

La mayoría de los métodos de aprendizaje no supervisado están destinadas a realizar un agrupamiento de los datos de entrada. En este tipo de análisis la información se agrupa según ciertas similitudes o características compartidas entre los diferentes tipos de entradas.

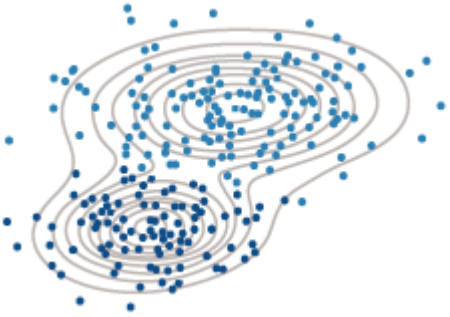


Figura 7: Técnicas no supervisadas de ML para agrupación de datos [10].

En este caso, cuando se quiere que la computadora aprenda a identificar a un gato, lo que se hará será introducirle una gran cantidad de imágenes, pero esta vez sin etiquetar. De este modo, la red neuronal comenzará a buscar por sí misma las características comunes de estas imágenes. Una vez se haya entrenado lo suficiente, la máquina contará con un número suficiente de parámetros característicos que le permitan identificar si una imagen dada representa a un gato o no.

2.2.3. Deep Learning

El deep learning es un tipo de machine learning en el que un modelo es capaz de aprender a agrupar una elevada cantidad de información. Esto se conseguirá mediante la construcción de una red neuronal que irá mejorando su desempeño en función del número de capas ocultas de redes neuronales y del tamaño de la información introducida para entrenar a dicha red.

El deep learning ha demostrado ser la mejor opción para desempeñar una amplia variedad de tareas, y la razón es su eficiencia y exactitud. Todos los diferentes tipos de dispositivos electrónicos han mejorado notablemente la implementación de su inteligencia artificial, incluyendo algoritmos de deep learning cada vez más eficientes. Tanto ha sido esta evolución que en ámbitos como la clasificación de imágenes se han llegado a conseguir mejores resultados que los que podría aportar cualquier ser humano. Los motivos que han hecho posible que este tipo de

técnicas puedan mejorar hasta conseguir este elevado grado de exactitud en tan poco tiempo podrían resumirse como:

- Fácil acceso a conjuntos masivos de datos clasificados gratuitos.
- Ordenadores con unidades de procesamiento cada vez más potentes, que permiten el procesamiento de grandes cantidades de información en pocas horas.
- Modelos pre-entrenados construidos por expertos que pueden ser re-entrenados realizando una transferencia de aprendizaje.

El deep learning es un un tipo de machine learning inspirado en la estructura neuronal del cerebro humano. Como puede verse en la Fig.8 estas redes neuronales se basan en un procesamiento de la información jerárquico a través de múltiples capas no lineales.

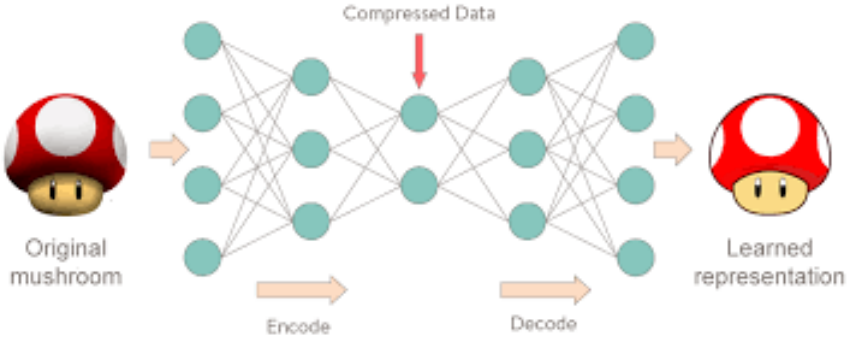


Figura 8: Red Deep learning [3].

Usando los datos de entrenamiento, la red busca patrones comunes en la información introducida. Cada capa de la red toma los datos de la capa anterior, la transforma y la vuelve transmitir, esta vez hacia la siguiente capa. Cada capa filtra la información, agrupándola según ciertas características y disminuyendo el número de parámetros necesarios para definirla. La red aumenta su complejidad cada vez que la información pasa de una capa oculta a otra. Cabe destacar que cada neurona aprende de forma independiente a partir de los datos de entrada, es decir, no se tiene control alguno sobre la forma en que se realiza dicho aprendizaje dentro de cada unidad neuronal.

Al ser un método de aprendizaje no supervisado, el deep learning es una buena opción cuando se quiere modelar un gran volumen de datos con el fin de la búsqueda de patrones. Esto es realmente útil ya que no es necesario conocer con exactitud los datos de entrada y salida con anterioridad.

2.2.4. Machine Learning vs Deep Learning

Aunque en ocasiones machine learning y deep learning aparecen como sinónimos no son lo mismo. El deep learning constituye un caso particular de machine learning. La principal diferencia entre ambos es que para el machine learning es necesario extraer manualmente las características relevantes de la información a analizar, mientras que con deep learning la red neuronal aprende por sí misma a partir de la información de partida sin ser previamente procesada.

Machine Learning	Deep Learning
Buenos resultados con pequeñas cantidades de información inicial.	Necesita gran cantidad de información inicial.
Rápido de entrenar	Necesita muchos recursos computacionales.
Necesita que se le introduzcan diferentes ejemplos de una categoría para conseguir buenos resultados.	Identifica y clasifica la información automáticamente.
Exactitud limitada.	La exactitud que puede obtener es ilimitada.

Tabla 1: Machine Learning vs Deep Learning [3].

2.3. Elección

El problema que se tratará en este trabajo, puede descomponerse en un problema de caracterización y su posterior clasificación. Es por ello que se intentará resolver en dos etapas. En una primera etapa se abordará la tarea de caracterización de los diferentes posibles estados del sistema. Para esto se utilizará una red neuronal que reducirá los datos de entrada para obtener una representación final bidimensional donde estén claramente diferenciados los estados de error del sistema. Una vez realizado el entrenamiento de la red neuronal se pasará a la clasificación de la información ya agrupada. La finalidad será conseguir que el sistema pueda autodiagnosticarse informando del tipo de fallo que se está produciendo.

Debido a la gran cantidad de información con la que habremos de tratar y al desconocimiento de la evolución exacta del sistema, para el proceso de caracterización se pueden descartar las técnicas de machine learning,puesto que la cantidad de datos que habría que introducir para entrenar a la red, así como el tiempo computacional requerido sería enormemente elevado. Además, como puede verse en la Fig.9 el error obtenido para este tipo de reconstrucciones por un proceso de deep learning será notablemente inferior al que podría obtenerse al intentar realizar la misma clasificación por un proceso de machine learning.

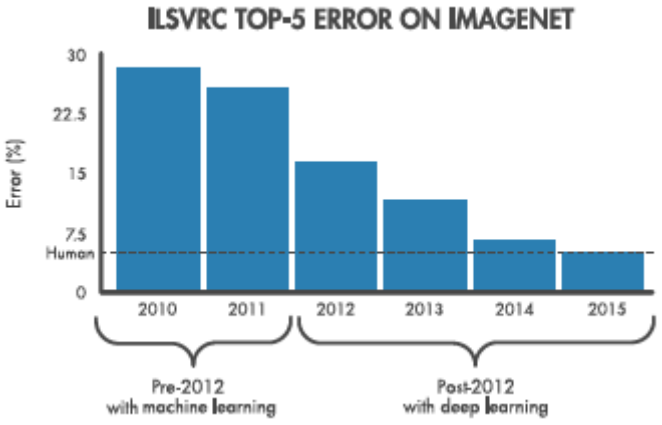


Figura 9: comparación del error obtenido con machine learning y deep learning [3].

Más concretamente, el tipo de arquitectura que se utilizará para implementar dicha red consistirá en una codificación mediante Autoencoders dispersos. Los autoencoders son una técnica de aprendizaje profundo no supervisado en el que dados una gran cantidad de ejemplos X que siguen una distribución completamente desconocida $D(X)$, el modelo tratará de aprender un modelo que reproduzca $D(X)$ posible.

2.3.1. Autoencoder

Puede decirse que un autoencoder consiste en una composición de dos redes neuronales diferentes: el codificador (encoder) y el decodificador (decoder) ambas unidas por un nexo común que será el espacio latente. Aunque hasta el momento en todas las secciones anteriores esta composición ha sido presentada como una única unidad es necesario considerar su naturaleza independiente para su correcto posterior estudio.

Los resultados obtenidos al final del decoder dependerán de sus tres componentes principales:

- Codificador: Es la primera parte de la red neuronal, aquí el modelo aprende a reducir la información de entrada, esto es, reduce su dimensión y la comprime para crear una nueva representación. Esta proceso es fundamental en el entrenamiento del autoencoder, ya que debe de crear muestras reducidas pero al mismo tiempo con la cantidad de detalles suficiente como para realizar una buena caracterización del problema. El diseño del encoder o codificador supondrá la parte inicial del presente estudio.
- Espacio latente: En ocasiones es denominado como cuello de la botella, ya que hace referencia a la representación más reducida del codificador. No obstante, esta muestra no va a ser siempre la misma, sino que variará cada vez que entrenemos el autoencoder, ya que esta última capa es solo una distribución de probabilidades. Es por ello que será necesario realizar numerosas pruebas e iteraciones para elegir el espacio latente más conveniente. Finalmente, esta última capa constituirá la entrada a la primera capa del decodificador.
- Decodificador: en el decodificador o decoder ser realiza el proceso inverso al realizado previamente en el codificador. A partir de la muestra reducida se reconstruye la información para intentar reproducir con la mayor exactitud posible los datos de partida. Por tanto, podría decirse que un autoencoder no es más que una forma determinada de compresión de datos con la menor pérdida de información posible.

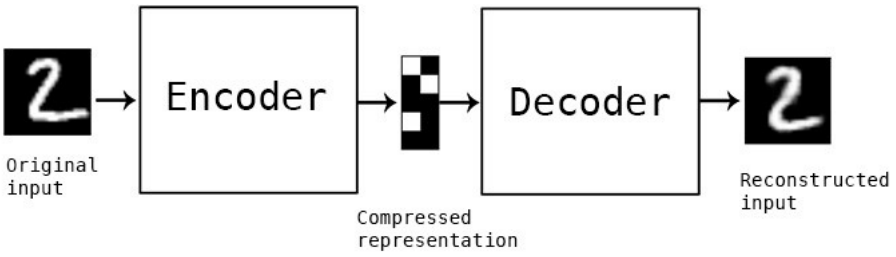


Figura 10: Reconstrucción de un autoencoder [2].

3. Desarrollo de las soluciones escogidas

De las técnicas citadas en el punto anterior, se decide usar el deep learning para la codificación de los datos de entrada y para la caracterización de los datos ya codificados. Por tanto, se hará uso del aprendizaje no supervisado. Esto ahorra la ardua tarea de clasificación previa al paso por el algoritmo.

Se decide utilizar un autoencoder para la codificación de los datos. Un autoencoder es una estructura de red neuronal que lo que intenta es codificar los datos de entrada haciéndolos pasar por un bloque de encoder para luego tratar de reproducirlos haciendolos pasar por un decoder. La implementación de este tipo de estructuras es especialmente sencilla en Matlab gracias al uso del toolbox Deep Learning [11].

Varios autoencoders pueden ser apilados para obtener una estructura más compleja que permitirá disminuir el tamaño de la muestra a cada paso por un bloque de encoder, permitiendo obtener, al final de los mismos, y antes de pasar por los decoders, una versión sintetizada de los datos de entrada. Esta versión será la que se utilice para la caracterización de los datos.

3.1. Preparación de los datos

3.1.1. Descripción de la bancada electromecánica

Los datos necesarios para entrenar la inteligencia artificial son extraídos de una bancada electromecánica la cual simula los diferentes estados del actuador real.

El banco de pruebas se basa en dos motores encarados y de características idénticas: el motor bajo prueba y el motor que actúa como carga. Estos motores se conectan mediante un tornillo y un reductor, constituyendo el banco de ensayos. Uno de los motores acciona el eje de entrada del reductor. El eje de salida del reductor acciona el tornillo, que a su vez, desplaza la pieza móvil. Los motores son dos SPMSM de 3 pares de polos, con un par nominal de 3,6 Nm, 230 Vac, y una velocidad nominal de 6000 rpm proporcionada por ABB Group. Los motores han sido accionados también por convertidores de potencia ABB modelo ACSM1.

El equipo de medición está enfocado al registro de vibraciones. El transductor del acelerómetro, se conecta a un sistema de adquisición PXIe 1062 suministrado por NI. La frecuencia de muestreo ha sido fijada en 20kS/s durante 1 segundo para cada experimento. El diagrama de la disposición experimental se muestra en la Fig. 11.

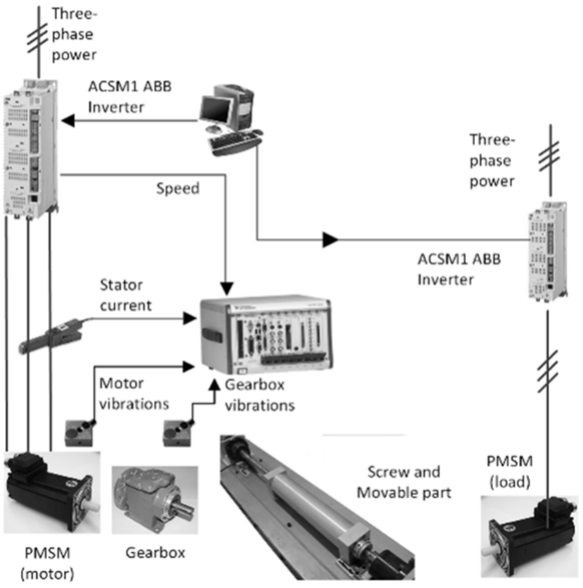
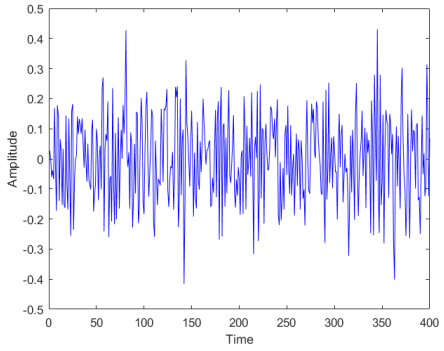


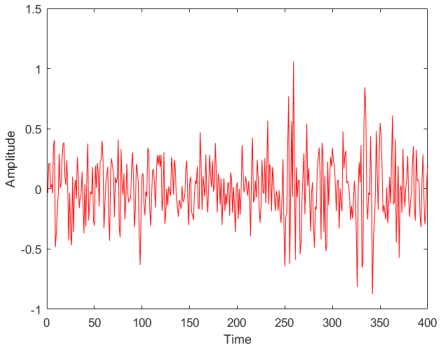
Figura 11: Bancada de pruebas

3.1.2. Muestra inicial

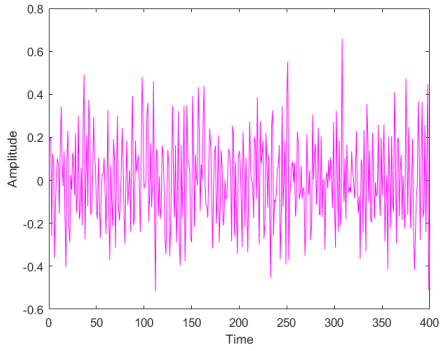
El formato que se tiene de los datos de partida son vectores de 2,72M de componentes que representan todas las amplitudes que puede adquirir la señal de vibración del sistema, dentro de un rango determinado de frecuencias. En estas señales la evolución de la amplitud viene dada por el tiempo. Se dispondrá de 5 vectores de igual dimensión que aportarán información para las cuatro condiciones de error ya mencionadas en 1.1 y un quinto que representará el estado sano del sistema. Se seleccionan muestras de cada vector de partida y se distribuyen en matrices de dimensión 400x20. Las 5 matrices resultantes se unen formando una única matriz de 400x100 con datos correspondientes a todos los estados posibles que podrán darse en el sistema. Como puede verse en las siguientes gráficas es muy complicado identificar características claras de cada estado, y las oscilaciones son bastante complicadas de diferenciar de un estado a otro.



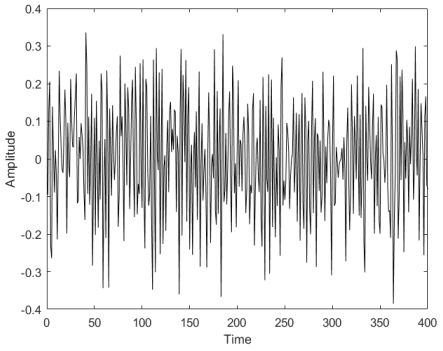
(a) Healthy condition.



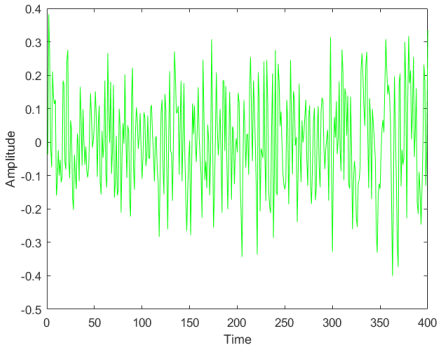
(b) Bearing fault condition.



(c) Demagnetization fault condition.



(d) Eccentricity fault condition.



(e) Gear fault condition.

Figura 12: Representación de los datos iniciales.

No obstante, al realizar los primeros ensayos de compresión y reconstrucción en el autoencoder se comprueba que el encoder no consigue realizar una buena diferenciación de los estados, y que por lo contrario todos los colores (cada uno correspondiente a un estado diferente) se muestran completamente mezclados. Esta representación por tanto haría imposible una correcta caracterización y por ende su posterior clasificación, por lo que se buscarán otras alternativas.

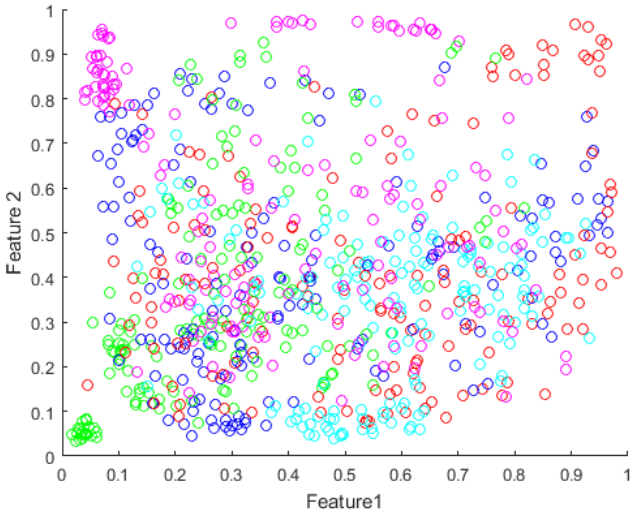


Figura 13: Reconstrucción de la salida de la matriz de datos inicial

Además, se calculó la varianza de estos ensayos. La varianza mide la diferencia entre el error del conjunto de pruebas y el error obtenido previamente durante el entrenamiento:

$$Varianza = Error_{pruebas} - Error_{entrenamiento} \tag{1}$$

$$Sesgo = Error_{entrenamiento} \tag{2}$$

Una varianza elevada implica que el autoencoder no aportará buenos resultados en la reconstrucción de datos. En este caso, de estos ensayos se observó que se daba un sobreajuste o sobreentrenamiento del encoder ya que, la varianza resultó considerablemente alta. Esto implica que la red no dispone de capacidad de adaptación suficiente como para recrear una señal de entrada distinta a la proporcionada durante el entrenamiento.

3.1.3. Re-parametrización

Como puede verse en la Fig.12 en esta representación es muy complicado diferenciar las diferentes amplitudes que adopta el sistema, esto es debido a que la amplitud varía constantemente con el tiempo, por lo que realizar la predicción de la señal frente al tiempo resulta realmente complicado si se quieren obtener buenos resultados. Por esto, será necesario realizar un procesamiento previo a los datos de vibración. Este procesamiento consistirá en la obtención de la transformada rápida de Fourier (FFT).

La transformada de fourier es una herramienta matemática utilizada para transformar señales definidas en el dominio del tiempo, pasando a quedar representadas en el dominio de la frecuencia. Cabe destacar, que tras aplicar la transformada de Fourier la información permanece exactamente igual, es decir, no existe pérdida de información, solo se modifica la forma en la que viene presentada. La expresión de la transformada de Fourier se define tradicionalmente como:

$$f(t) = \int_{infty}^{-infty} F(s)e^{2\pi st}dt \tag{3}$$

Donde f representa la ecuación continua en el dominio del tiempo y F la función una vez transformada en el dominio de la frecuencia.

Como puede deducirse de la representación de mostrada en la Fig. 12 no es frecuente encontrar señales puramente periódicas en el procesado de señales digitales. Además un sistema digital, como son las computadoras digitales, no pueden operar con funciones continuas, es por ello que es necesario trabajar con una señal muestreada. La Transformada Discreta de Fourier (DFT) es una variación de la trasformada de Fourier tradicional que nos permite realizar la transformación de dominios pero a partir de una función discreta. Sin embargo la DFT requiere que la señal de entrada sea puramente periódica ya que sólo puede analizarse un periodo de la misma. Esto es algo que es muy complicado de encontrar en problemas reales y que además haría de el cálculo de factores de estas series un proceso computacionalmente muy costoso.[12])

Sea *x(n)* una señal aperiódica discreta en el tiempo, la DFT puede definirse como:

$$X(k) = \sum_{n=0}^{N-1} x_n e^{-j2\pi \frac{n}{N}} \qquad k = 0, ..., N-1 \tag{4}$$

Donde *X(k)* es un conjunto de números complejos. La evaluación directa de la señal con esta fórmula necesitaría N^2 operaciones aritméticas, lo cual es un número considerablemente elevado. Debido a la complejidad que conlleva la DFT se decide utilizar otro tipo de algoritmo mucho más rápido que aporta resultados igualmente buenos, esto es la Transformada Rápida de Fourier (FFT). El número de operaciones aritméticas necesarias con este tipo de algoritmo será *NlogN*. Es decir, el número de operaciones aritméticas será considerablemente reducido conrespecto a la DFT. Es por esto que este algoritmo permitirá calcular la transformada de Fourier discreta pero con un coste de tiempo mucho mejor. Además es especialmente útil en el tratamiento de señales con ruido (incluso cuando este no es detectable por el ojo humano).

3.1.4. Muestra de datos final

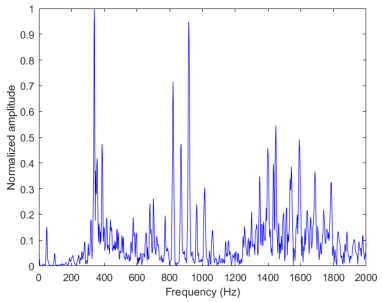
Una vez realizada la transformada rápida de Fourier se dispone a realizar una nueva matriz de datos. En esta ocasión se ampliará la dimensión de las muestras de datos. Las matrices consistirán en espectros de 2000Hz representados en 820 puntos de datos. Cada punto será la representación de la amplitud del sistema para una frecuencia dada dentro del rango de 2000Hz.

En el caso de la matriz de entrenamiento se dispondrán 80 muestras de cada estado con 820 puntos de datos cada muestra. Esto constituirá una matriz final de dimensión 820x400. Del mismo modo las matrices con la que se realizarán la validación y las posteriores pruebas estará constituida de 30 columnas (esto es, 6 muestras por estado)[14].

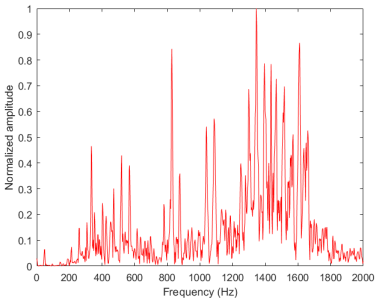
$$A = \begin{pmatrix} a_{1\,1} \cdots a_{1\,80} & a_{1\,81} \cdots a_{1\,160} & a_{1\,161} \cdots a_{1\,240} & a_{1\,241} \cdots a_{1\,320} & a_{1\,321} \cdots a_{1\,400} \\ a_{2\,1} \cdots a_{2\,80} & a_{2\,81} \cdots a_{2\,160} & a_{2\,161} \cdots a_{2\,240} & a_{2\,241} \cdots a_{2\,320} & a_{2\,321} \cdots a_{2\,400} \\ a_{3\,1} \cdots a_{3\,80} & a_{3\,81} \cdots a_{3\,160} & a_{3\,161} \cdots a_{3\,240} & a_{3\,241} \cdots a_{3\,320} & a_{3\,321} \cdots a_{3\,400} \\ \vdots \ddots \vdots & \vdots \ddots \vdots & \vdots \ddots \vdots & \vdots \ddots \vdots & \vdots \ddots \vdots \\ \underbrace{a_{820\,1} \quad a_{820\,80}}_{1^{er}\ Estado} & \underbrace{a_{820\,81} \quad a_{820\,160}}_{2^{o}\ Estado} & \underbrace{a_{820\,161} \quad a_{820\,240}}_{3^{er}\ Estado} & \underbrace{a_{820\,241} \quad a_{820\,320}}_{4^{o}\ Estado} & \underbrace{a_{820\,321} \quad a_{820\,400}}_{5^{o}\ Estado} \end{pmatrix}$$

Figura 14: Estructura de la matriz de datos de entrenamiento.

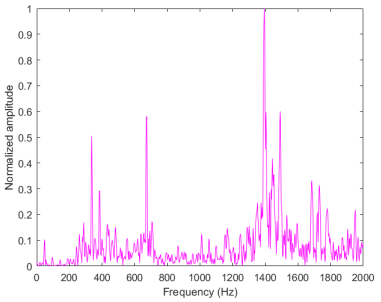
Finalmente, una vez realizada esta nueva representación puede verse en la Fig.15 como puede diferenciarse de forma mucho mas clara la evolución de la amplitud del sistema.



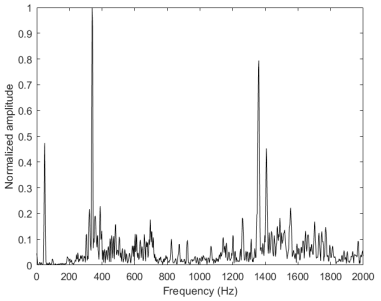
(a) Healthy condition.



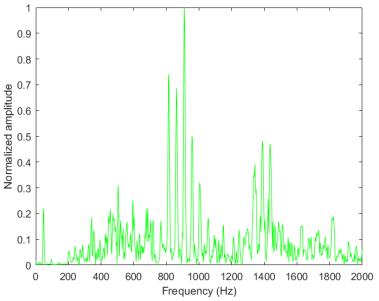
(b) Bearing fault condition.



(c) Demagnetization fault condition.



(d) Eccentricity fault condition.



(e) Gear fault condition.

Figura 15: Representación de los datos de partida tras aplicar FFT

En este caso, y comparando con los mismos datos de la Fig. 12, se observa que las señales son fácilmente distinguibles a simple vista, lo que implica que también lo será para la red neuronal. Además, para facilitar aún más el trabajo, se han escalado los datos normalizando la amplitud de tal forma que se encuentre en un rango entre cero y uno en cualquiera de los casos.

3.1.5. Gestión del set de datos

La gran cantidad de datos y la complejidad del modelo requiere tiempos de entrenamiento muy largos. Como tal, es típico separar los datos en:

- Datos de entrenamiento
- Datos de prueba
- Datos de validación

Una vez el modelo esté entrenado, lo siguiente será validarlo. Esto se realizará con los datos de validación (que no habrán sido utilizados para el entrenamiento previo). Sobre el set de validación se procederá a correr el algoritmo y a evaluar los resultados obtenidos. Esta evaluación tendrá como objetivo descartar la posibilidad de sobrentrenamiento del sistema. Hay que tener en cuenta la posibilidad de que el modelo funcione bien para el set de entrenamiento y no para el set de validación. Por ello cabe resaltar que se volverá a realizar el entrenamiento hasta que el modelo se ajuste bien a las dos particiones de los dato usados hasta ahora. Todo esto con el propósito de ganar confianza en nuestro modelo. Siguiendo la distribución típicamente usada se ha elegido la siguiente configuración del set de datos para el diseño del modelo:

Datos de entrenamiento	Datos de prueba	Datos de validación
70 %	15 %	15 %

Una vez comprobado que el modelo ha sido entrenado correctamente mediante los datos de validación, se pasará a la prueba del sistema. En este caso se usarán los datos de prueba de los cuales se obtendrán lo resultados finales del estudio del modelo (véase la sección 4 del presente trabajo).

3.2. Composición de la red neuronal

Una vez se tiene una visión global sobre la arquitectura y funcionamiento de los autoencoders se realizará el diseño del modelo que será usado para estudio de este proyecto. Las decisiones de diseño de una red neuronal pasan por definir parámetros tales como el número de capas de la red, función de activación o pesos iniciales.Se realizará el diseño de un autoencoder disperso, ya que esta es la configuración utilizada por el toolbox del programa de computación Matlab, en el cual ha sido implementado este trabajo.

Como ya ha sido explicado con anterioridad en el apartado 2.3.1un autoencoder estará formado por tres partes conectadas entre sí: encoder, decoder y espacio latente. Las dos primeras constituyen redes neuronales en sí mismas y están diseñados para que tengan una arquitectura en espejo, es decir, son simétricas una respecto a la otra. El espacio latente constituirá el nexo común de ambas redes.

Debido a que se desea tener una representación bidimensional para la posterior clasificación de los datos, la información debe ser comprimida (codificada) a través de las capas de encoder de la red. Dado que los datos de entrada vienen representados por una matriz de 820 filas (ver sección 3.1.4), comprimirla hasta tener una representación bidimensional es equivalente a obtener a la salida del último encoder una matriz con 2 filas y el mismo número de columnas. Además, el número de filas corresponde con el número neuronas en la capa.

Comprimir los datos en una sola capa se torna brusco, y los resultados no serán satisfactorios. Se decide pues, tener varias capas ocultas, en concreto tres. La determinación del número de neuronas en cada capa se explica más adelante en la sección 3.4. Cada capa que se añade a la red tiene el coste computacional de entrenar un autoencoder completo con una capa oculta. Es por ello que el número elegido de capas es el mencionado, ya que como puede observarse en en el anexo A añadir más capas no consigue una mejora significativa la reconstrucción de las características, de hecho el error cuadrático medio obtenido en estos casos es incluso mayor. Además añadir más capas ocultas supone un aumento del coste computacional considerable.

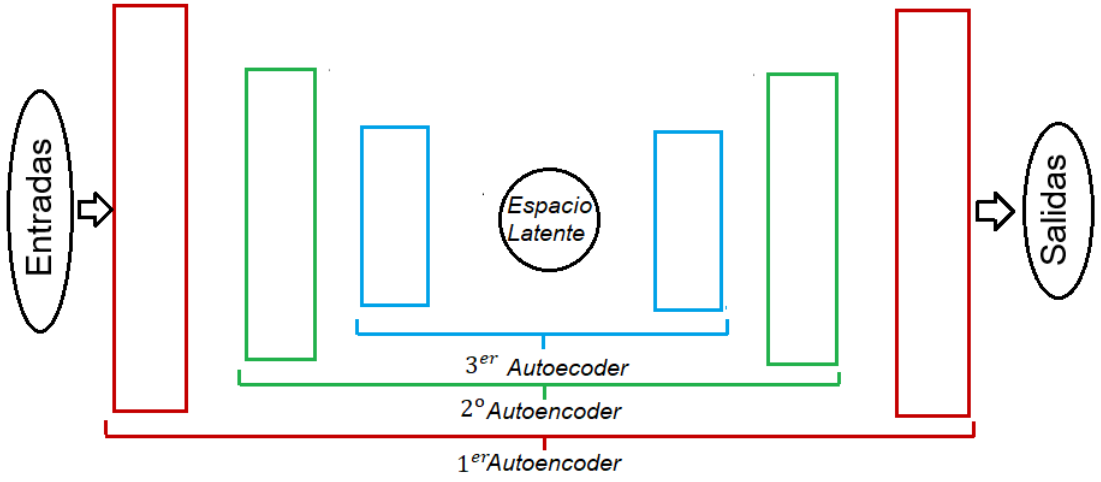


Figura 16: Arquitectura del autoencoder implementado

En las siguientes subsecciones se desarrollará la estructura general de una red neuronal, su implementación en Matlab y finalmente el estudio paramétrico que definirá las características de la red que se entrenará finalmente.

3.2.1. Estructura general un encoder

El principal objetivo de esta estructura será la de aprender una representación de los datos de entrada pero tras una gran reducción de dimensión. Como puede verse en la Fig.17 existen una serie de parámetros que

determinarán la forma en que la red neuronal aprenderá a captar elementos característicos de la entrada inicial.

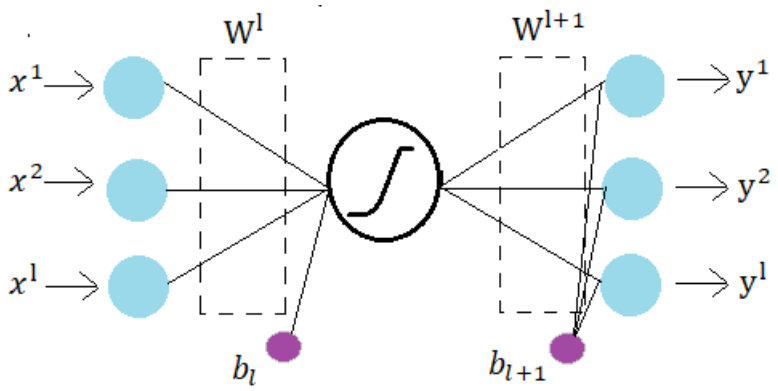


Figura 17: Parámetros que influyen en una red neuronal

El primer parámetro que podemos observar es el peso que se le asocia a las conexiones entre neuronas para cada capa oculta de la red neuronal. El peso representará una medida de la complejidad de la red y en la Fig.17 viene denotada por la variable $W_{i,j}$. Por otro lado la variable b estará asociada al sesgo que como ya ha sido definido en 3.1.2 representa el error asociado a la fase de entrenamiento. Y finalmente justo en el centro de la red neuronal encontramos la función de transferencia asociada al espacio latente. Para este caso la función de transferencia será una función sigmoidal definida por la ecuación:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

Donde x serán las entradas del autoencoder.A partir de estos parámetros se calculará la función de activación denotada por la variable a. Así la salida del encoder representado en la figura podría calcularse a partir de las siguientes funciones:

$$z^{(1)} = W^{(1)} \cdot x + b^{(1)} \tag{6}$$

$$a^{(1)} = f(z^{(1)}) \tag{7}$$

$$z^{(2)} = W^{(2)} \cdot x^{(2)} + b^{(2)} \tag{8}$$

$$y = z^{(2)} \tag{9}$$

Estas ecuaciones nos permitirán ajustar el entrenamiento del encoder para la obtención de la representación reducida de los datos iniciales. Esta representación será por tanto la entrada del decoder. Aunque este ejemplo solo muestra un encoder con una capa oculta, este proceso se repetirá tantas veces como capas ocultas haya. Para el entrenamiento del autoencoder se utilizará además el método de backpropagation, cuyo objetivo es el de minimizar el error obtenido a la salida de la red.

3.2.2. Algoritmo backpropagation

En las secciones anteriores se ha descrito el funcionamiento de las neuronas y como pueden usarse para resolver tareas de regresión o clasificación muy simples. A partir de ahí, se ha visto también como pueden juntarse para crear una estructura más compleja, una red neuronal, con la que modelar información mucho más elaborada. También se ha mostrado su estructura geométrica y la influencia de diversos parámetros que modificarán la capacidad de predicción de la red. El problema viene pues cuando hay que decidir cómo modificar estos parámetros con el fin de obtener buenos resultados. Estos podrían ser adaptados manualmente a partir de innumerables iteraciones, pero obviamente haría el proceso demasiado lento y potencialmente más impreciso. Además la base de la inteligencia artificial es precisamente, conseguir que la red neuronal aprenda a modificar de forma autónoma estos parámetros.

Para conseguir el objetivo de que una red neuronal aprenda automáticamente se creó un algoritmo, en base al cual la red neuronal mediante un proceso iterativo ajustara de forma autónoma las variables que influyen

de una u otra forma en la salida del sistema. Esta técnica partiría del estudio del descenso del gradiente en el que la estrategia consiste en modificar los parámetros de una regresión lineal evaluando el error del modelo en un punto determinado con el fin de calcular las derivadas parciales en dicho punto. Así, se consigue un vector de direcciones que indica la pendiente hacia donde el error se incrementa, lo que se conoce como gradiente. Simplemente tomando la dirección opuesta se obtendría el camino por el que se iría disminuyendo el error. No obstante, este proceso es sencillo para el caso de unas pocas neuronas que no calculan más que regresiones lineales, ya que en estos casos no se tienen más que unas pocas de variables. Cuando se tratan de redes neuronales, hay demasiados parámetros que influyen en la salida del sistema, y por tanto, el cálculo del gradiente se dificulta demasiado.

El objetivo del problema será observar como cambia la función de coste, dada por la expresión:

$$E(W, b) = \frac{1}{2}||y - x||^2 \tag{10}$$

Siendo x una de las entradas del sistema e y la salida del mismo, calculada a partir las ecuaciones 6-9 y que por tanto, será función de los pesos y el sesgo de cada uno de los enlaces por los que pasa la señal hasta llegar a la salida.

Para facilitar la compresión se va a desarrollar el siguiente ejemplo. En la Fig.18 se presenta el proceso de estudio de un estudiante el cual recorrerá toda la red hasta llegar a la salida. Esta salida será el resultado del examen al que deberá presentarse y ver si lo supera o de lo contrario suspende. Sin embargo, como puede verse, hay múltiples factores que influyen en este resultado, como serían las horas de estudio, el profesor que imparte la asignatura, las horas de sueño o incluso la asignatura de la que se trata. Todas estas opciones introducen multitud de caminos por el que llegar al final del proceso.

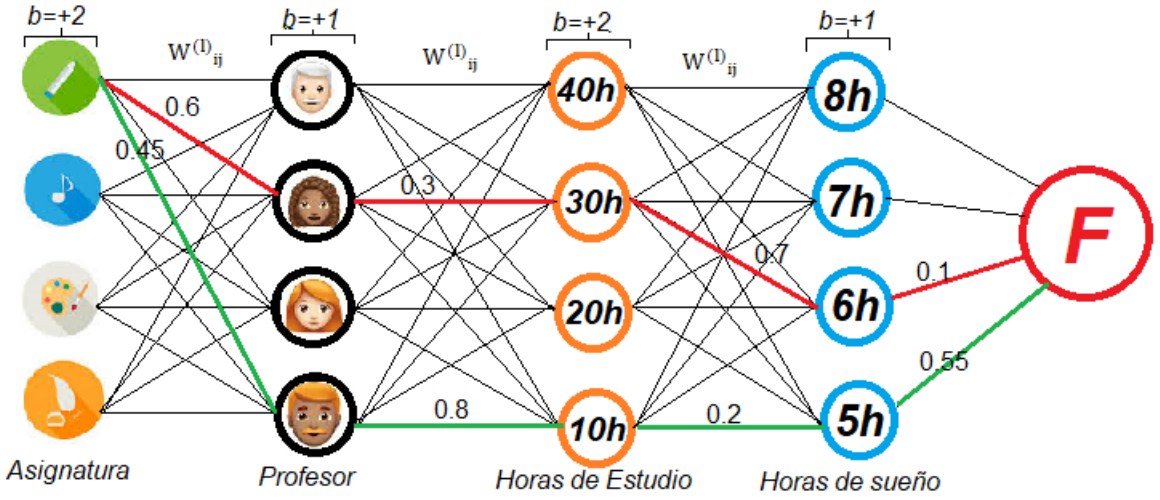


Figura 18: Influencia del peso en el cálculo de una salida

La expresión que nos aportaría la salida de la red siguiendo el camino rojo y verde respectivamente sería pues: (Ver 8).

$$y = 0,1(0,7(0,3(0,6x + 2) + 1) + 2) + 1 = 0,0126x + 1,312 \tag{11}$$

$$y = 0,55(0,2(0,8(0,45x + 2) + 1) + 2) + 1 = 0,0396x + 2,386 \tag{12}$$

Puede comprobarse que para una misma entrada x las salidas y serán diferentes, y puede deducirse que lo mismo pasaría con el resto de combinaciones posibles. Entonces, al querer estudiar qué factor es el responsable del suspenso del alumno la tarea sería extremadamente complicada, ya que los caminos hasta llegar al suspenso son innumerables. Es aquí donde aparece la técnica de backpropagation. Este algoritmo permitirá estudiar qué

grado de responsabilidad tiene cada una de las fases en el resultado final, pero esta vez se comenzará desde el final de la red y esta será recorrida hasta su inicio en sentido inverso. Así por ejemplo, si se comprueba que el profesor que imparte la asignatura no tiene una gran influencia en el resultado del examen, podrá dictaminarse que tampoco lo hacen ninguna de las capas previas, es decir, la asignatura, y podrá centrarse el estudio únicamente en las capas posteriores a la capa "profesor".

Así pues, se partirá de la capa final de la red (el suspenso del alumno), donde se evaluará el error final obtenido. Con este error, se avanzará hacia a la capa previa: "horas de sueño" donde distribuiremos la influencia que han tenido las diferentes neuronas en este error. Es decir, si durante la semana el alumno ha dormido un 70 % de los días 5h, esta neurona tendrá una responsabilidad mayor que las demás. Y así se irá avanzando progresivamente hasta llegar al final de la red. Una representación más visual puede verse en la Fig.19 donde se ha representado la responsabilidad de cada factor en el resultado final variando el tamaño de la "F".

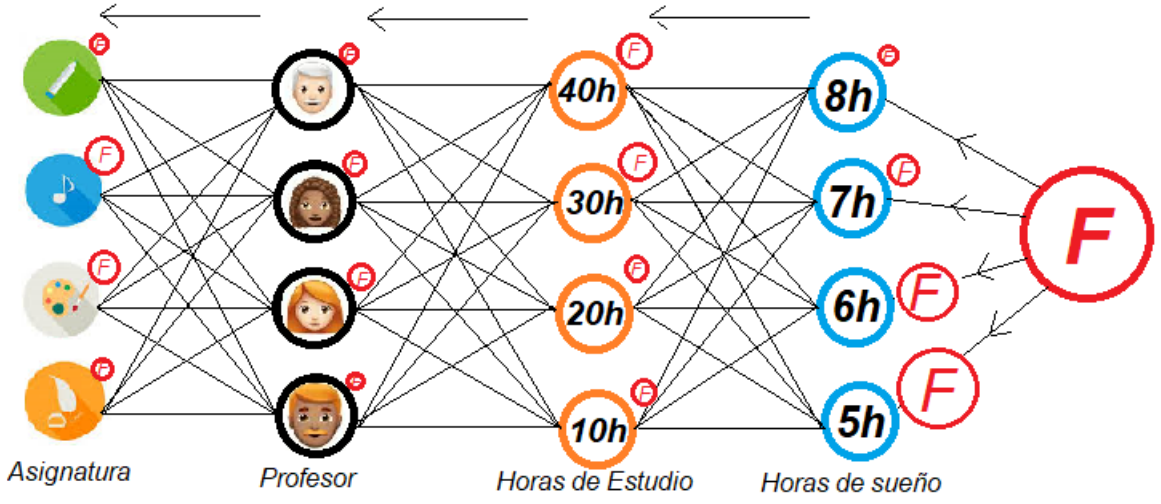


Figura 19: Responsabilidad de cada unidad neuronal

De esta forma, según la responsabilidad de cada neurona, el encoder podrá ajustar automáticamente los parámetros que influencian el cálculo de la salida de la red. Como observación, decir que además con esta técnica no sería necesario realizar la exploración de todas las combinaciones posibles para llegar al final de la red, ya que como se mencionó en la sección 3.2.1 las neuronas disponen de una función umbral que provocará la activación selectiva de ciertas conexiones determinadas para cada entrada. Con la técnica de retropropagación, sólo se estudiarán aquellas conexiones que han sido activadas para una salida concreta del sistema.

Este algoritmo es implementado por un elaborado desarrollo matemático, el cual no ha sido incluido ya que es calculado automáticamente por la función `trainAutoencoder`. Pero ha querido incluirse esta explicación cualitativa [13] ya que se ha considerado interesante para una mejor comprensión del trabajo. No obstante un desarrollo más detallado de estas ecuaciones puede encontrarse en: la referecia [14].

3.2.3. Autoencoder con dispersión

La principal funcionalidad de un autoencoder disperso es evitar la especialización excesiva de la red neuronal. Esto ocurre cuando todas las neuronas aprenden o retienen las mismas características. Si todas las neuronas aprendieran por igual, al final lo que ocurriría es que se especializarían en un única entrada (la memorizarían) pero cuando se les introdujera una señal diferente la reconstrucción tendría un error demasiado elevado. Es por esto que interesa que cada neurona retenga unas ciertas características de la señal introducida, pero que estas características difieran de las que aprendan el resto de neuronas. Para obtener este propósito, se introduce un parámetro que durante el proceso de aprendizaje, active selectivamente ciertas neuronas concretas para cada cada iteración.

Se dice que una neurona está activa si el valor de su salida es cercano a 1 y por lo contrario, será 0 si

la neurona está en un estado inactivo. Por lo tanto, el propósito de un encoder con dispersión será limitar el número de neuronas activas para que se mantengan inactivas la mayor parte del tiempo.

Considerando la función de activación a ya definida anteriormente en 3.2.1, y siendo x una determinada entrada, la activación media de una neurona j podrá definirse como:

$$\hat{\rho} = \frac{1}{m}\sum_{i=1}^{m}[a_j(x^{(i)}] \tag{13}$$

El objetivo será cumplir:

$$\hat{\rho} = \rho \tag{14}$$

Siendo ρ el parámetro de dispersión, que convendrá mantener en valores cercanos a cero (del orden de $\rho = 0{,}05$). En otras palabras, se buscará que la activación media de mayoría de las neuronas sea cercana a 0. De este modo, se conseguirá que cada neurona se especialice en el reconocimiento de un cierto número de características concretas pero que ignore el resto. Así se conseguirá que la red sea mucho más adaptable y pueda interpolar buenos resultados para cualquier entrada. Este término de dispersión será introducido en la función de costes, y constituirá otro de los parámetros que será necesario ajustar para optimizar la actuación del autoencoder a diseñar.

Además, como ha podido verse en la sección 3.1.3, las señales que se presentan, al provenir de un sistema relativamente complejo traen consigo mucho ruido. Se entiende ruido como todo tipo de información que está adherida a los datos de entrada pero que no aporta información útil para la resolución del problema. Con el fin de obtener unos mejores resultados, será necesario eliminar gran parte de estas perturbaciones. Otra de las ventajas que aporta un autoencoder disperso es su capacidad para eliminar el ruido de estas señales, facilitando el aprendizaje de la red.

3.2.4. Parámetros de una red neuronal

Una vez explicados los elementos de una red neuronal así como su funcionamiento, es necesario presentar qué parámetros pueden ser modificados para diseñar una red neuronal propia.

Empezando por el tamaño, pueden definirse tanto el número de capas ocultas como el número de neuronas que cada capa contiene. El número de neuronas en una capa permite decidir el tamaño de la codificación de los datos que estás hacen. Esto es que si se desea tener una representación bidimensional de los datos, es decir, por cada muestra reducir las variables a dos, esto se hará mediante una capa con dos neuronas, donde cada neurona recibirá los datos de todas las variables (el número de variables a las que reaccione dependerá de la dispersión aplicada) y a la salida de la capa solo se tendrán dos, una por neurona. Aumentar el número de capas permitirá hacer una reducción (aumento) más paulatina del número de variables, reduciendo la posible pérdida de información tras el paso por las neuronas. Como se ya se ha visto, un aumento en exceso del número de capas no mejora el aprendizaje.

Pasando al comportamiento de las neuronas, este puede modificarse de varias formas. Por una parte, puede decidirse la función de activación y por otra la función de transformación que aplica a las variables que entran a la misma. Además, en el caso de redes dispersas, se tendrá un parámetro de proporción de la dispersión, el cual indicará el número de datos a los que reacciona cada neurona.

Aunque el objetivo es minimizar una función de costes por debajo de un valor umbral, también es posible decidir el número de iteraciones que, como máximo, puede utilizar la red para aprender, llamado número máximo de Epochs. Este parámetro evita entrar en un bucle infinito en el que el error se quede estancado en un valor por encima del umbral. Si además guardamos el error en cada Epoch, se podrá ver su evolución para actuar en consecuencia cambiando alguno de los otros parámetros en caso de que el valor objetivo no fuera alcanzado.

Finalmente, puede elegirse la función objetivo de la red así como el algoritmo de entrenamiento. La función objetivo suele ser buscar el mínimo (o máximo) de una función, por ejemplo, el error cuadrático medio. Esta función puede tener términos añadidos que tengan en cuenta los pesos de las variables en la diferentes capas así como la dispersión, en cuyo caso, cada término lleva un parámetro de regularización asociado (como es el caso

del parámetro de regularización L-2). El algoritmo de entrenamiento será aquel que varíe el peso asociado a cada variable y neurona buscando avanzar hacía el mínimo (máximo) de la función. El algoritmo más extendido para redes neuronales es el de backpropagation, ya explicado anteriormente (sección 3.2.2).

3.3. Implementación en Matlab

En Matlab, gracias a la utilización del toolbox Deep Learning (disponible a partir de la versión 2015 de Matlab), es posible implementar diferentes tipos de redes neuronales de manera sencilla. A pesar de ello, se tiene control de todos los parámetros mencionados en las secciones anteriores. A continuación se detalla como implementar los dos tipos de estructuras utilizadas en este trabajo: el autoencoder y la capa de clasificación.

3.3.1. Autoencoder

Las funciones básicas que implementa Matlab para la utilización de una estructura de tipo autoencoder son:

```
X = dataset;
autoencoder = trainAutoencoder(X);
XReconstructed = predict(autoencoder,X);
mseError = mse(X-XReconstructed);
```

Donde:

- `trainAutoencoder(X)` será la función encargada de entrenar a la red neuronal (es decir, calcular los pesos de cada neurona) a partir de los datos contenidos en X. Los diferentes parámetros vistos en las secciones anteriores pueden ser modificados (se desarrolla a continuación).
- `predict(autoencoder,X)` devolverá la salida de hacer pasar los datos X a través del autoencoder (red ya entrenada). Al ser un autoencoder, debe ser lo más parecido posible a los datos originales.
- `trainAutoencoder(mse(X-XReconstructed))` devuelve el error cuadrático medio de los datos reconstruidos con el autoencoder en comparación con los originales.

Dentro de la función `trainAutoencoder()`, los parámetros que han sido definidos para una mejor caracterización del problema han sido definidos en la tabla [2]. No obstante, no todos los parámetros serán modificados para la creación de la red deseada. Esto se verá en la sección 3.4. Además una descripción más detallada de los parámetros y su utilización puede encontrarse en la referencia de Matlab [15].

En Matlab, un autoencoder está formado por una capa de entrada (encoder), una capa oculta (hidden layer) y una capa de salida (decoder). Como se ha comentado en la subsección 3.2.1, se desea crear una red con 3 capas ocultas. Esto es posible en Matlab apilando autoencoders. Para ello, es necesario utilizar, además de las funciones ya mencionadas, las siguientes:

- `enc1 = encode(autoenc1,X)` devuelve la transformación que sufre el set de datos X en la capa de encoder del autoencoder (ya entrenado) `autoenc1`.
- `dec1 = decode(autoenc1,enc1` devuelve la reconstrucción de los datos (transformados previamente por el encoder) `enc1` en la capa de decoder del autoencoder (ya entrenado) `autoenc1` .
- `stakednet = stack(autoenc1, autoenc2, autoenc3)` devuelve una red neuronal resultado de la apilación las capas de encoder de los autoencoders que se dan como parámetros de entrada. En último lugar también puede añadirse una capa de softmax (explicado más adelante en esta sección).
- `y =stakednet(X)` devuelve los datos de salida de introducir X en la red stackednet. Es el equivalente a la función predict descrita anteriormente en esta sección.

Parámetro	Descripción	Valor por defecto
hiddenSize	Número de neuronas en la capa oculta.	10
EncoderTransferFunction	Función de transferencia para las neuronas de la capa de entrada.	signoide
DecoderTransferFunction	Función de transferencia para las neuronas de la capa de salida.	signoide
MaxEpochs	Número máximo de entrenamientos.	1000
L2WeightRegularization	Regularizador del termino de los pesos en la ecuación de costes.	0.001
LossFunction	Ecuación de costes.	Mínimos cuadrados dispersos
SparsityProportion	Proporción de datos a los que se quiere que la neurona reaccione.	0.05
SparsityRegularization	Regularizador del término de dispersión de la ecuación de costes.	1
TrainingAlgorithm	Algoritmo de aprendizaje.	Descenso del gradiente conjugado
ScaleData	Escala los datos de entrada al rango de las funciones de transferencia.	true
UsseGPU	Indicador para usar la GPU para entrenar la red.	false

Tabla 2: Parámetros de entrada de la función `trainAutoencoder` de Matlab

Un ejemplo de utilización y funcionamiento detallado puede encontrarse en la referencia de Matlab [16]. De esta forma, se tiene un esquema como el de la figura 20, que es equivalente a tener un solo autoencoder con 3 capas ocultas. Finalmente, es igualmente posible representar la estructura en Matlab utilizando la función `view(stakednet)`.

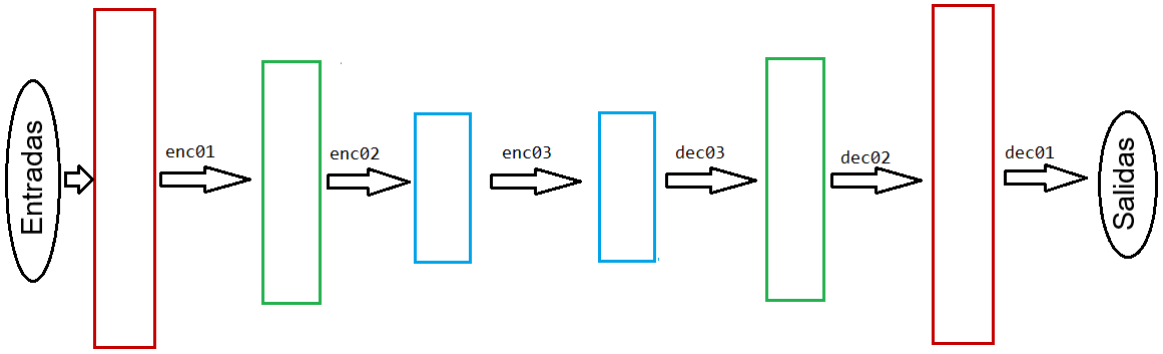


Figura 20: Autoencoders apilados

3.3.2. Red de patrones (capa de clasificación)

La clasificación de los diferentes datos en Matlab se hace a partir de una red especializada en clasificación la cual se crea con la función `patternnet()` (Ver Fig 23). Como parámetros de entrada a la función, se tiene el número de capas ocultas, la función de entrenamiento y la de performance.

El entrenamiento de la misma, se realiza con la función `train(net,X,T)`.En este caso, además de los datos de entrada X (los datos ya comprimidos por el autoencoder en nuestro caso), es necesario proporcionar una matriz T la cual indica la clase a la que pertenece cada una de las columnas de X. En este caso, los únicos parámetros que pueden ser modificados son `MaxEpochs`, `TrainingFcn`.

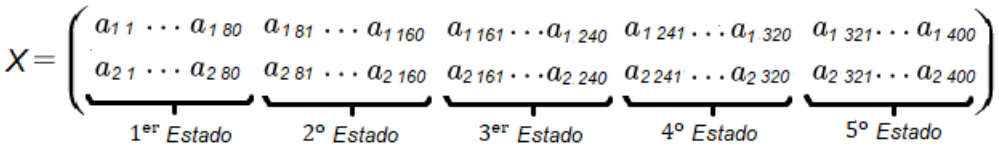


Figura 21: Salida del encoder

Donde las componentes $a_{i,j}$ serán las componentes relativas a la reducción bidimensional del encoder.

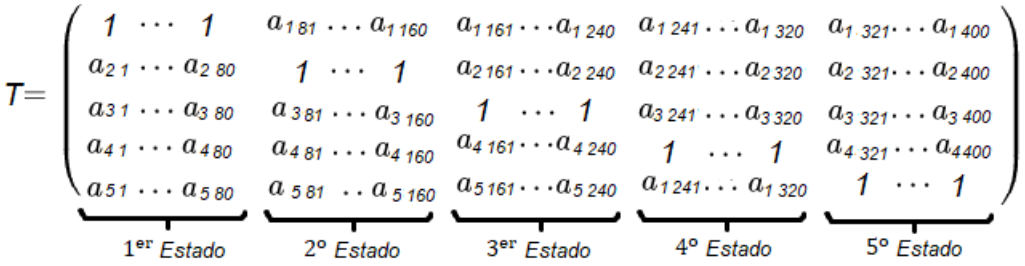


Figura 22: Matriz de clasificación de estados

En este caso, todas las componentes $a_{i,j}$ de la matriz T serán igual a 0, de tal forma, que las componentes con valor igual a la unidad, indicarán el estado al que pertenece cada columna de datos.

Finalmente, para mostrar la matriz de confusión, basta con usar la función `plotconfusion(T,Y)`, donde Y es el resultado de usar la red patternnet.

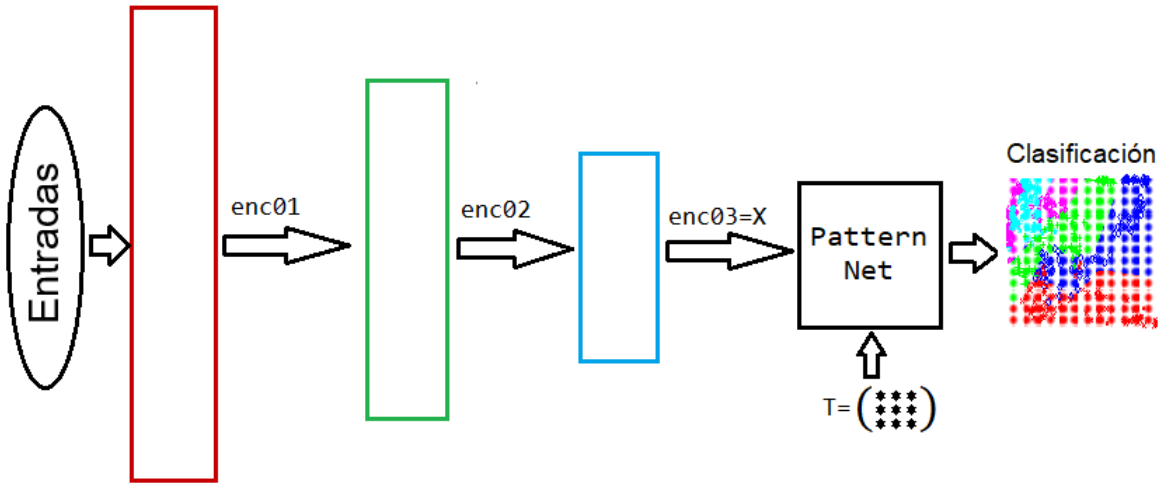


Figura 23: Clasificación usando una red patternnet

3.4. Estudio paramétrico de los bloques del autoencoder

En esta sección se presentarán los valores finales decididos para los parámetros del autoencoder ya definidos en la sección 3.3. Esta sección del trabajo podría dividirse en dos partes. En primer lugar se definirán los parámetros como el número máximo de entrenamientos, la dispersión y los pesos, además de la función de transferencia utilizada tanto para el encoder como el decoder. En la segunda parte se figurarán estos valores y se realizará de nuevo un proceso iterativo variando el número de neuronas de cada capa oculta.

Para el primer grupo de parámetros se comenzó inicialmente utilizando los valores aportados por la literatura [3] y que han sido reflejados en la tabla 2. Se realizaron varias iteraciones, en las que se observó la capacidad de

la red neuronal del encoder para realizar una representación en dos dimensiones de la información introducida donde se zonificara los diferentes estados de fallo. Con ayuda de los tutores del presente trabajo se fueron modificando los parámetros mencionados hasta conseguir los siguientes valores finales, que fueron fijados para el resto del desarrollo del proyecto.

Parámetro	**Valor**
hiddenSize	A determinar en la siguiente etapa
EncoderTransferFunction	signoide
DecoderTransferFunction	signoide
MaxEpochs	400
L2WeightRegularization	0.001
LossFunction	Mínimos cuadrados dispersos
SparsityProportion	0.00005
SparsityRegularization	0.00005
TrainingAlgorithm	Descenso del gradiente conjugado
ScaleData	true
UseGPU	false

Tabla 3: Selección de los parámetros finales

Una vez congelados los valores de los parámetros del autoencoder se procede a un nuevo estudio iterativo para determinar el número de neuronas en cada capa oculta. Como ya se ha visto en la sección 3.2 el autoencoder que se está desarrollando en este estudio constará de tres autoencoders apilados, por lo que habrá que definir un número de neuronas en la capa oculta para cada uno de estos autoencoders. Como lo que se busca es realizar una reducción de dimensionalidad de los datos introducidos, el número de neuronas de cada encoder irá disminuyendo respecto al número de neuronas del encoder anterior. Se probarán las siguientes configuraciones:

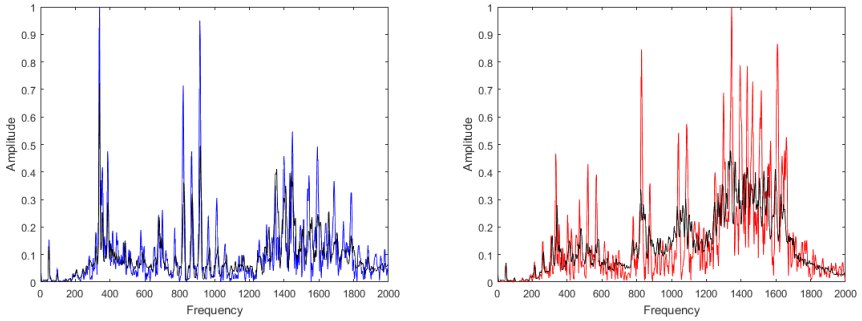
- 1^{er} Encoder: 600, 550, 500, 450, 400 ó 350 neuronas.
- $2^{\underline{o}}$ Encoder: 250, 200, 150, 100 ó 50 neuronas.
- 3^{er} Encoder: 75, 50, 25, 20 ó 15 neuronas.

La iteración se realizó fijando el número del segundo y tercer encoder a 150 y 25 neuronas respectivamente. Seguidamente se realizaron 10 pruebas para cada una de las configuraciones posibles del primer encoder. De cada prueba se extrajo el error cuadrático medio y se calculó el error cuadrático medio promedio de las 10 iteraciones para cada una de las configuraciones. Se eligió la configuración que aportó el error cuadrático medio promedio más bajo. El estudio realizado sobre la evaluación del error cuadrático medio ha sido incluido en el apéndice C.

Una vez fijado el valor de 550 neuronas en el primer encoder y determinados todos los parámetros de del mismo, se volvieron a realizar 10 iteraciones para cada una de las posibilidades ya definidas del segundo encoder. Seleccionada la configuración óptima del segundo encoder, se repitió el proceso con los valores del tercer encoder. Finalmente la configuración seleccionada resultó ser:

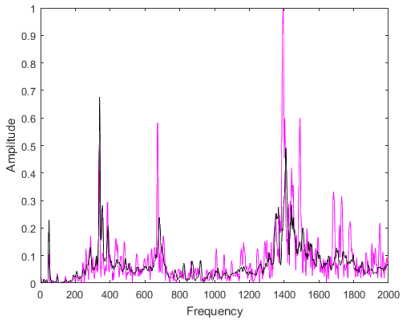
- 1^{er} Encoder: 550 neuronas.
- $2^{\underline{o}}$ Encoder: 250 neuronas.
- 3^{er} Encoder: 15 neuronas.

En las siguientes figuras 24, se muestra la reconstrucción (en negro) de la señal de entrada del autoencoder con la mencionada configuración final. Puede verse, que obviando valores extremos que seguramente sean causados por el ruido existente en la señal, el sistema consigue realizar una reconstrucción notablemente buena de la entrada. El error promedio obtenido para esta configuración fue de 0,00754.

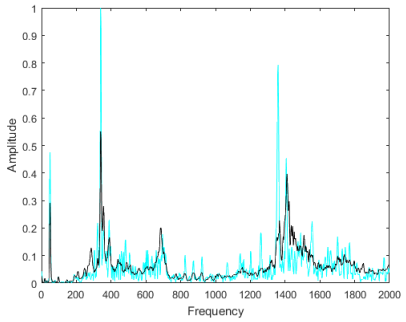


(a) Healthy condition.

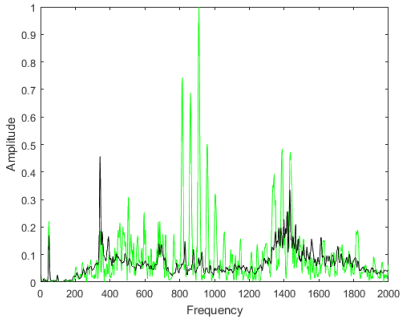
(b) Bearing fault condition.



(c) Demagnetization fault condition.



(d) Eccentricity fault condition.



(e) Gear fault condition.

Figura 24: Representación de la reconstrucción de las señales

3.5. Selección de parámetros de la red neuronal de clasificación

En cuanto a la selección de parámetros para la red neuronal de clasificación podemos hacer dos divisiones: los parámetros de configuración de la propia red, y por otro lado, los parámetros de la función de entrenamiento.

Para la clasificación, se utilizó una red neuronal simple de una capa oculta de cinco neuronas con una función sigmoidea como función de activación. En la Fig.4 pueden verse los valores finales que se han fijado para el diseño de la red neuronal de clasifiación.

Parámetro	Valor
Tamaño de la capa oculta	5
Función de activación	signoide
LossFunction	cross-entropy loss

Tabla 4: Parámetros de la red de clasificación

En cuanto a la función de entrenamiento de esta red neuronal cabe mencioar que el método de enteramiento que se ha utilizado es el backpropagation (véase la sección 3.2.2). También se ha definido el número máximo de Epochs, y el máximo de fallos admisibles en la validación como puede comprobarse en la siguiente figura 5.

Parámetro	Valor
Algoritmo de aprendizaje	backpropagation
Max Epochs	500
Número máximo de fallos en validación	1.500
Gradiente mínimo	1e-10

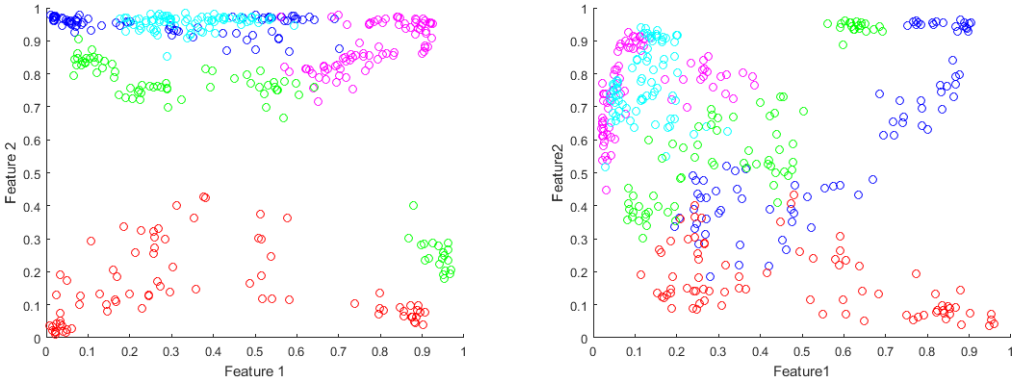
Tabla 5: Parámetros de la función de entrenamiento para la red de clasificación

.

4. Resumen de los resultados

A continuación se mostrarán los resultados correspondientes a la configuración paramétrica seleccionada en la sección 3.4 para cada autoencoder constituyente de la red neuronal. Se ha decidido representar los resultados correspondientes a las dos pruebas que arrojaron los mejores errores cuadráticos medios en el proceso de iteración ya descrito en la sección anterior. Estos corresponden a la prueba 6 y 10 de la configuración seleccionada (550, 250 y 15 neuronas para uno de los 3 encoders respectivamente).

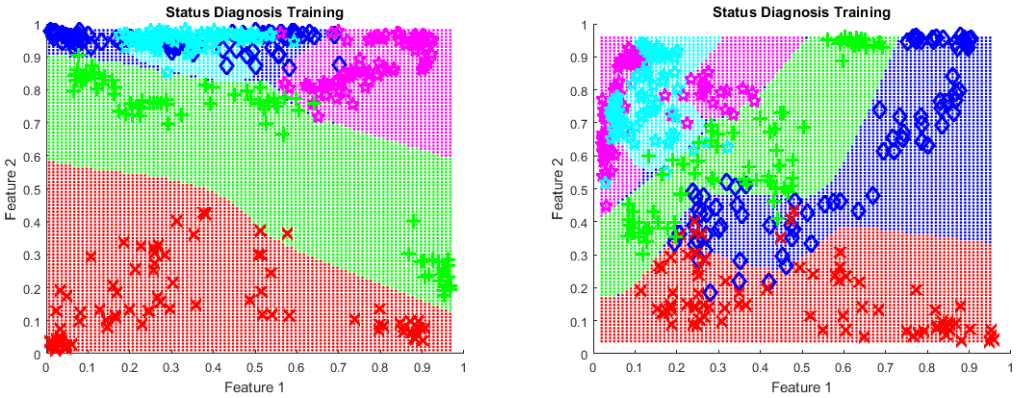
En primer lugar en las figuras 25 se muestra la representación en dos dimensiones de la salida del tercer autoencoder. Así puede verse como la red neuronal ha aprendido una representación de alto nivel de los datos, suficientemente buena como para poder distribuir los diferentes estados que adopta el sistema electromecánico. Puede apreciarse además como las señales que son similares permanecen cercanas en el espacio. Esto podría presentar algunos problemas a la hora de clasificación de las señales, pero como se comprobará un poco más adelante, los resultados obtenidos son suficientemente buenos.



(a) Prueba 6. (b) Prueba 10.

Figura 25: Representación 2D del decoder

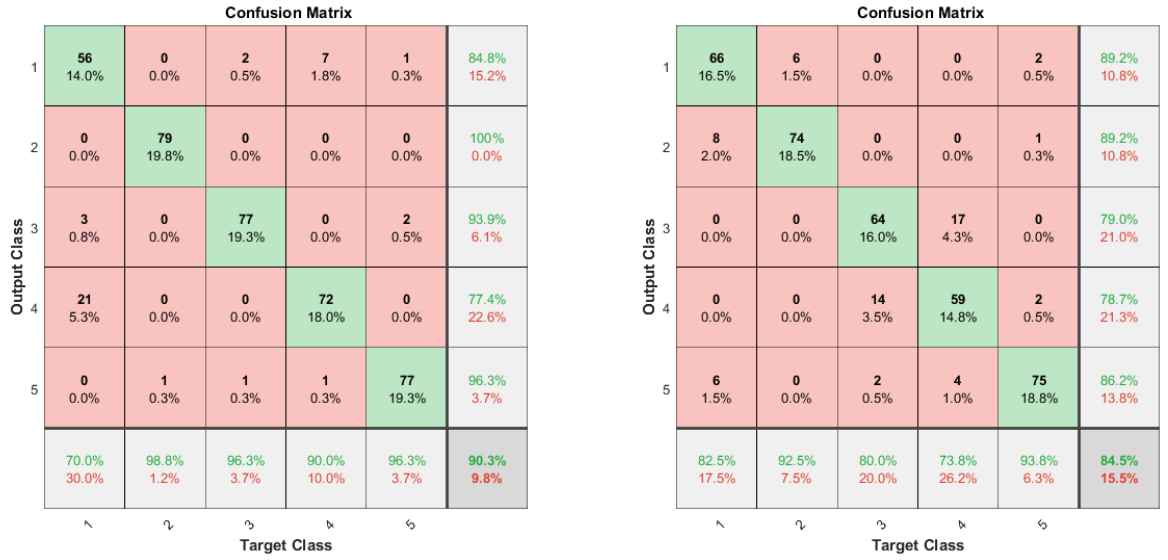
Con estas dos representaciones en dos dimensiones del espacio latente se procede a realizar la clasificación de los datos. Como ya se ha explicado en la sección 3.3, esto se hace a través de una red neuronal de clasificación la cual tendrá 5 capas ocultas. Para su entrenamiento, la función encargada recibirá como entrada la representación bidimensional de los estados, así como la posición de los mismos en la matriz de datos y la función de entrenamiento (proporcionada por Matlab). Con este procedimiento, se obtendrán las siguientes figuras, que muestran una zonificación bastante clara según los estados de fallo del sistema.



(a) Prueba 6. (b) Prueba 10.

Figura 26: Clasificación softmax prueba.

Por último, para ver como de buenos son los resultados se extraerá la matriz de confusión. Esta matriz mostrará la probabilidad de pertenencia de cada condición a su clase correspondiente. Esta matriz será calculada con la función `plotconfusion()` de Matlab.



(a) Prueba 6.

(b) Prueba 10.

Figura 27: Matrices de confusión

La matriz de confusión representa la probabilidad que hay de que el sistema consiga identificar el estado que se está dando en el sistema. Las filas representan cada una de las clases que la red ha predicho mientras que las columnas la clase real. De esta forma, los elementos de la diagonal son aquellos que han sido correctamente predichos, no siéndolo así los que se encuentran fuera. Puede verse como los resultados obtenidos son considerablemente buenos para la clasificación de un sistema tan complejo como el que se le ha introducido. Para las pruebas 6 y 10 se obtendría un una correcta clasificación en el 90,3 % y un 84,5 % de media de los casos respectivamente.

Sin embargo, se torna necesario analizar cada estado por separado. La última fila y columna de cada matriz contiene métricas del porcentaje de fallo para cada una de las clases. La última fila representa el porcentaje de casos de cada clase que la red ha identificado correctamente y el porcentaje que no, respecto al número total de datos de esa determinada clase. Esto también es denominado tasa de verdadero positivo (en verde) y falso negativo (en rojo) respectivamente. Por otra parte, la última columna representa el porcentaje de acierto y error para las muestras que la red ha predicho pertenecer una clase concreta. A estos porcentajes también se les denomina tasa de predicción positiva (en verde) y tasa de predicción falsa (en rojo).

De esta forma en la prueba 6, con el mejor porcentaje total, puede observarse que un 30 % de las muestras del estado "Healthy" no han sido predichas como tal, teniendo un 26,2 % de esas muestras que han sido predichas como correspondientes al estado "Eccentricity". El resto de los estados poseen todos una tasa de falso negativo inferior al 10 %, teniendo el mínimo para el estado "Bearing" con un 1,2 %. Centrándonos en la tasa de predicción positiva, se puede resaltar que ninguna muestra es falsamente predicha como correspondiente al estado de "Bearing" y las que mayor tasa tienen son el estado "Healthy" (15,2 %) y "Eccentricity" (22,6 %).

En la prueba 10, se observan tasas notablemente peores tanto de falso negativo, donde el mínimo es 6,3 % (estado "Gear") y el máximo 26,2 % (estado "Eccentricity"), como de predicción falsa, teniendo un error mínimo del 10,8 % (estados "Healthy" y "Eccentricity" y un máximo del 21,3 % (estado "Eccentricity"). En ambos casos, los mayores errores están asociados al estado "Eccentricity". En la prueba 6 las muestras del estado "Healthy" son las únicas que la red confunde con el estado "Eccentricity" y en la prueba 10, los estados "Demagnetization" y "Eccentricity" se confunden el uno con el otro en un porcentaje parecido (17,5 % y 21,2 % respectivamente).

5. Conclusiones

En el presente trabajo, se ha han implementado técnicas de inteligencia artificial para el diagnóstico automático de un posible estado de error en un actuador electromecánico de un dispositivo hiper-sustentador. El trabajo puede dividirse en tres partes principales: preparación de los datos de entrada, compresión de los datos, clasificación automática y extracción de resultados.

Debido a la complejidad del sistema, los datos proporcionados necesitaban un tratamiento previo a la utilización del deep learning. Esta fase es crucial, puesto que tener o no buenos resultados depende mucho de tener un set de datos limpios (o bien tratados). Debido a ello, es necesario una reparametrización de los datos de partida, obteniéndose su representación en función de la frecuencia en vez de en el tiempo.

Una vez los datos estaban listos, se pasó a la fase de entrenamiento, donde se hizo un estudio paramétrico de las diferentes opciones ofrecidas por Matlab para así obtener la configuración óptima del autoencoder que permita reproducir con la mayor precisión los datos iniciales. Esto significa que se llega a una versión comprimida de mayor calidad. Este proceso es lento, debido a que cada test (con 3 capas ocultas) tenía una duración de alrededor tres minutos y medio, que multiplicado por las distintas configuraciones que se propusieron y el número de iteraciones necesarias por cada configuración, se convirtieron en una gran inversión de tiempo. Este coste computacional alto es compensado por la cantidad de tiempo y dinero que se ahorraría en el caso de conseguir un algoritmo lo suficientemente preciso en la detección del estado del actuador.

Una vez entrenada la red, se observa que por más que se varíen los parámetros de entrada, resulta imposible que la representación comprimida de los estados estén en su totalidad perfectamente diferenciados. Esto se debe a la complejidad del problema real, que hace que obtener una clasificación clara sea complicado.

Finalmente, gracias a otra red neuronal, se hace el trabajo de clasificación para los dos casos cuya reconstrucción tenía un error menor. En ambos casos se ha obtenido una precisión del 90,3 % y el 84,5 %. Estos valores, teniendo en cuenta el sistema real y la normativa general asociada a la seguridad de los componentes eronáuticos es baja.

En conclusión, se ha cumplido el objetivo de crear una inteligencia artificial que autodiagnostica el estado del actuador con una cierta precisión y de forma potencialmente más eficiente que el diagnóstico que podría hacer cualquier operario de forma manual. Queda así demostrado el gran potencial que presentan las técnicas de deep learning, y cómo pueden mejorar el desempeño de una gran variedad de tareas.

5.1. Planificación y programación del trabajo futuro propuesto

En primer lugar se podrían estudiar diferentes técnicas de tratamiento de señales para la eliminación de ruido. Esto permitiría conseguir una versión más limpia de los datos de partida y por tanto una clasificación con un porcentaje de éxito mucho mayor.

Por otra parte, una forma de mejorar el entrenamiento es disponiendo de una gran cantidad de datos de partida. Esto permite tener más datos con los que entrenar y testear el algoritmo, con el fin de asegurar que la precisión del diagnóstico sea la deseada. Por lo que una posible mejora sería repetir el entrenamiento con un número de entradas mucho mayores. Por otro lado, se ha podido observar que hay estados que provocan un error considerablemente mayor que los demás. Una solución sería aumentar los datos de entrenamiento para estos estados concretos con el fin de conseguir una red con mayor especialización en la caracterización de estos estados conflictivos.

Además un posible estudio de interés sería el análisis del número de encoders óptimo en una arquitectura de autoencoders apilados. Ya que como ha podido verse en la sección 3.2, el que haya muchas capas ocultas no implica necesariamente que los resultados obtenidos vayan a ser satisfactorios.

Referencias

1. P., Abigail. *Fotos que marcaron la historia*. 2019. **urlalso**: https://hotpoptoday.com/post-5541263989/8/?dai=3TPW21kXptcXzYPESnSRaC&fbclid=IwAR2LUaxOH9FJT2C7lR7H1Gx2qlzKoZrrfIL2HDMYmmNow9QHnuQ_oFakdws.
2. BADR, Will. *¿Qué es un Autoencoder?* 2019. **urlalso**: https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.
3. MATHWORKS. *Deep Learning*. 2019. **urlalso**: https://es.mathworks.com/.
4. *Historia de las Redes Neuronales*. 2019. **urlalso**: http://www.itnuevolaredo.edu.mx/takeyas/apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/RNA/Redes%20Neuronales2.pdf.
5. *Terminator 3: La rebelión de las máquinas*. 2003. **urlalso**: https://www.filmaffinity.com/es/film477986.html.
6. MURCIA, Departamento de Nuevas Tenologías y Contaminación de Atmósferas de la Universidad de. *Equivalencia entre redes artificiales y biológicas*. 2019. **urlalso**: https://www.um.es/LEQ/Atmosferas/Ch-VI-3/F63s4p3.htm.
7. OLIVERA, Oscar García-Olalla. *Redes Neuronales artificiales: Qué son y cómo se entrenan*. 2019. **urlalso**: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i.
8. MATHWORKS. *Getting Started with Machine Learning*. 2019. **urlalso**: https://es.mathworks.com/.
9. MATHWORKS. *Applying Supervised Learning*. 2019. **urlalso**: https://es.mathworks.com/.
10. MATHWORKS. *Applying Unsupervised Learning*. 2019. **urlalso**: https://es.mathworks.com/.
11. MATHWORKS. *Deep Learning Toolbox*. 2019. **urlalso**: https://es.mathworks.com/products/deep-learning.html.
12. *Transformada Rápida de Fourier*. 2019. **urlalso**: https://es.wikipedia.org/wiki/Transformada_r%C3%A1pida_de_Fourier.
13. CSV, Dot. *¿Qué es una Red Neuronal?: Backpropagation*. 2019. **urlalso**: https://www.youtube.com/watch?v=eNIqz_noix8.
14. CASTELLANOS, Rosa Martínez Álvarez. *Análisis de las máquinas "Sparse Autoencoders" como extractores de características*. 2017. **urlalso**: http://repositorio.upct.es/handle/10317/6591.
15. MATHWORKS. *Implementación en Matlab de la función"trainAutencoder"*. 2019. **urlalso**: https://es.mathworks.com/help/deeplearning/ref/trainautoencoder.html.
16. MATHWORKS. *Ejemplo de utilización de apilación de autoencoders para clasificación*. 2019. **urlalso**: https://es.mathworks.com/help/deeplearning/examples/train-stacked-autoencoders-for-image-classification.html).

Índice de figuras

Índice de Tablas

Apéndices

A. Representación 2D con 4 y 5 capas ocultas

Capas ocultas	MSE	Coste computacional (min)
4	0.0080	4:00
5	0.0080	4:30

Tabla 6: Error cuadrático medio y coste computacional para diferentes números de capas ocultas

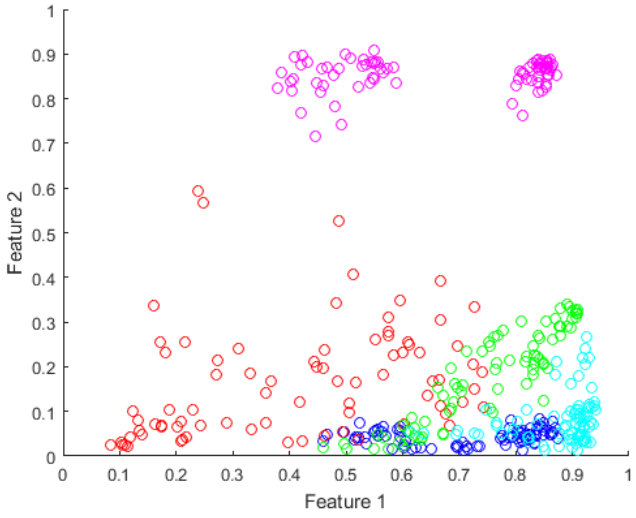


Figura 28: Representación salida del encoder con 4 capas ocultas

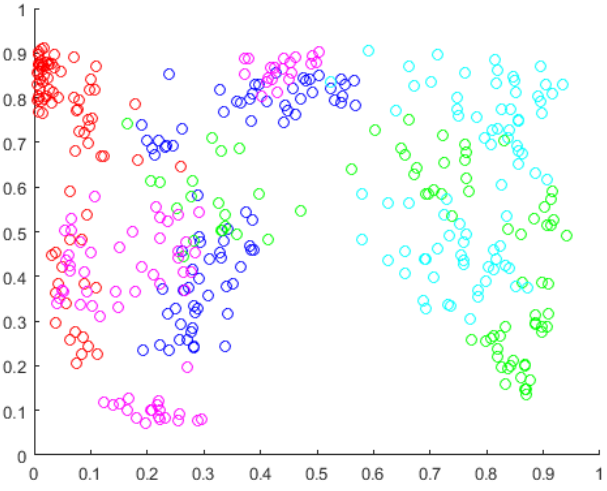


Figura 29: Representación salida del encoder con 5 capas ocultas

B. Códigos de Matlab

B.1. Entrenamiento de autoencoders apilados

```
function [Xreconstructed, enc04, mseError]= funcionautoencoders(TrainData,hl1,hl2,hl3)
% Esta funcion entrena una red neuronal de tres capas ocultas y devuelve la reconstruccion de
    los datos, la codificacion de los mismos y el error cuadratico medio de la reconstruccion.
% Parametros de entrada:
    % TrainData : Datos de partida que se usaran para entrenar la red neuronal.
    % hl1, hl2, hl3 : Numero de neuronas en la primera, segunda y tercera capa oculta.

    %% Entrenar red
    autoenc0 = trainAutoencoder(TrainData,hl1,...
       'MaxEpochs',400,...
       'SparsityProportion',0.4,...
       'DecoderTransferFunction','logsig',...
       'SparsityRegularization',0.00005,...
       'L2WeightRegularization',0.00005,...
       'EncoderTransferFunction','logsig');
        enc01 = encode(autoenc0,TrainData);

        autoenc1 = trainAutoencoder(enc01,hl2,...
       'MaxEpochs',400,...
       'SparsityProportion',0.4,...
       'DecoderTransferFunction','logsig',...
       'SparsityRegularization',0.00005,...
       'L2WeightRegularization',0.00005,...
       'EncoderTransferFunction','logsig');
        enc02 = encode(autoenc1,enc01);

        autoenc2 = trainAutoencoder(enc02,hl3,...
       'MaxEpochs',400,...
       'SparsityProportion',0.4,...
       'DecoderTransferFunction','logsig',...
       'SparsityRegularization',0.00005,...
       'L2WeightRegularization',0.00005,...
       'EncoderTransferFunction','logsig');
        enc03 = encode(autoenc2,enc02);
        autoenc3 = trainAutoencoder(enc03,2,...
       'MaxEpochs',400,...
       'SparsityProportion',0.4,...
       'DecoderTransferFunction','logsig',...
       'SparsityRegularization',0.00005,...
       'L2WeightRegularization',0.00005,...
       'EncoderTransferFunction','logsig');
    %apilar decoders
        enc04 = encode(autoenc3,enc03);
        dec04 = predict(autoenc3,enc03);
        dec03=decode(autoenc2,dec04);
        dec02=decode(autoenc1,dec03);
        dec01=decode(autoenc0,dec02);
        Xreconstructed = dec01;
        Train1 = TrainData;
        mseError = mse(Train1-Xreconstructed)
end
```

B.2. Entrenamiento de capa de clasificación

```
function [net,tr,y] = entrena_nn(TrainingExperiments3Level,TrainingTargets3Level,
    hiddenLayerSize);


x = TrainingExperiments3Level';
t = TrainingTargets3Level';
%hiddenLayerSize = 2; %las pruebas son con 8

clear net tr
net = patternnet(hiddenLayerSize);

net.divideParam.trainRatio = 0.7;  %70/100;
net.divideParam.valRatio =  0.15; %15/100;
net.divideParam.testRatio = 0.15; % 15/100;

% Train the Network
net.trainFcn = 'trainrp';
net.trainParam.epochs=500;
net.trainParam.max_fail=1500;
net.trainParam.min_grad=1e-10;

[net,tr] = train(net,x,t);

% Test the Network
y = net(x);

figure, plotconfusion(t,y)

end
```

B.3. Ejemplo de proceso completo

```
clear all
clc

load('TrainData.mat');

[Xreconstructed, enc04, mseError] = funcionautoencoders(TrainData , 550 , 250 , 15)

InputData = enc04;

% labels
T_1(1:80,1) = 1;
T_1(81:160,2) = 1;
T_1(161:240,3) = 1;
T_1(241:320,4) = 1;
T_1(321:400,5) = 1;

% Number of neurons in the hidden layer
hiddenLayerSize=5;
[net,tr] = entrena_nn(InputData',T_1,hiddenLayerSize);

% Grid range
x = min(InputData(1,:)):0.01:max(InputData(1,:));
y = min(InputData(2,:)):0.01:max(InputData(2,:));

[X, Y] = meshgrid(x,y);
X = X(:);
Y = Y(:);
grid = [X Y];
size_grid = size(grid);

grid = grid';

nn_class = net(grid);        %% neural network classifier
y1 = nn_class;
y2=round(y1');

for i=1:1:size(y2,1)
    if y2(i,1)==1
        Z(i,1)=1;
    elseif y2(i,2)==1
        Z(i,2)=2;
    elseif y2(i,3)==1
        Z(i,3)=3;
    elseif y2(i,4)==1
        Z(i,4)=4;
    elseif y2(i,5)==1
        Z(i,5)=5;
    end
end

%% Class colors
[foo , class] = max(Z');
class = class';
```

```
colors = ['b.'; 'r.';'m.';'c.';'g.'];

figure
hold on
for j = 1:5
  thisX = X(class == j);
  thisY = Y(class == j);
  plot(thisX, thisY, colors(j,:));
end

P=InputData';
L=size(InputData,2);

for j=1:L
    if T_1(j,1)==1
    plot(P(j,1),P(j,2),'d','LineWidth',2,'MarkerEdgeColor','b','MarkerSize',10);
    elseif T_1(j,2)==1
    plot(P(j,1),P(j,2),'x','LineWidth',2,'MarkerEdgeColor','r','MarkerSize',10);
    elseif T_1(j,3)==1
    plot(P(j,1),P(j,2),'p','LineWidth',2,'MarkerEdgeColor','m','MarkerSize',10);
    elseif T_1(j,4)==1
    plot(P(j,1),P(j,2),'p','LineWidth',2,'MarkerEdgeColor','c','MarkerSize',10);
    else
    plot(P(j,1),P(j,2),'+','LineWidth',2,'MarkerEdgeColor','g','MarkerSize',10);
    end
end

xlabel('Feature 1','FontSize',11);
ylabel('Feature 2','FontSize',11);
title('Status Diagnosis Training')
```

C. Evolución de error cuadrático medio en el estudio del número de neuronas en cada capa oculta

Neuronas HL 1	Neuronas HL 2	Neuronas HL 3	Error por iteración		Error medio de la configuración
600	150	25	0.0080	0.0080	0.0081
			0.0083	0.0083	
			0.0078	0.0080	
			0.0077	0.0084	
			0.0081	0.0084	
550	150	25	0.0086	0.0080	0.00799
			0.0078	0.0080	
			0.0079	0.0080	
			0.0079	0.0079	
			0.0078	0.0080	
500	150	25	0.0082	0.0082	0.00804
			0.0081	0.0081	
			0.0080	0.0081	
			0.0080	0.0079	
			0.0078	0.0080	
450	150	25	0.0086	0.0080	0.00805
			0.0080	0.0081	
			0.0082	0.0078	
			0.0081	0.0082	
			0.0079	0.0076	
400	150	25	0.0080	0.0082	0.00808
			0.0078	0.0079	
			0.0080	0.0080	
			0.0083	0.0084	
			0.0080	0.0082	
350	150	25	0.0082	0.0084	0.00806
			0.0080	0.0083	
			0.0079	0.0079	
			0.0080	0.0079	
			0.0082	0.0078	

Tabla 7: Desglose del error en cada una de las configuraciones propuestas para la primera capa oculta.

Neuronas HL 1	Neuronas HL 2	Neuronas HL 3	Error por iteración		Error medio de la configuración
550	250	25	0.0071	0.0078	0.00774
			0.0079	0.0077	
			0.0079	0.0078	
			0.0077	0.0079	
			0.0077	0.0079	
550	200	25	0.0081	0.0079	0.0079
			0.0077	0.0079	
			0.0076	0.0082	
			0.0078	0.0081	
			0.0078	0.0081	
550	150	25	0.0078	0.0077	0.00794
			0.0082	0.0077	
			0.0083	0.0078	
			0.0080	0.0080	
			0.0079	0.0077	
550	100	25	0.0079	0.0079	0.008
			0.0078	0.0081	
			0.0083	0.0081	
			0.0080	0.0080	
			0.0081	0.0078	
550	50	25	0.0079	0.0080	0.00804
			0.0082	0.0080	
			0.0080	0.0085	
			0.0079	0.0083	
			0.0077	0.0079	

Tabla 8: Desglose del error en cada una de las configuraciones propuestas para la segunda capa oculta.

Neuronas HL 1	Neuronas HL 2	Neuronas HL 3	Error por iteración		Error medio de la configuración
550	250	75	0.0078	0.0079	0.00788
			0.0079	0.0078	
			0.0079	0.0078	
			0.0077	0.0081	
			0.0079	0.0080	
550	250	50	0.0079	0.0080	0.00781
			0.0076	0.0077	
			0.0077	0.0076	
			0.0078	0.0078	
			0.0082	0.0078	
550	250	25	0.0073	0.0076	0.00773
			0.0079	0.0074	
			0.0082	0.0075	
			0.0081	0.0080	
			0.0078	0.0075	
550	250	20	0.0076	0.0077	0.00764
			0.0077	0.0080	
			0.0078	0.0079	
			0.0074	0.0074	
			0.0076	0.0073	
550	250	15	0.0074	0.0081	0.00754
			0.0078	0.0074	
			0.0074	0.0071	
			0.0080	0.0075	
			0.0076	0.0071	

Tabla 9: Desglose del error en cada una de las configuraciones propuestas para la tercera capa oculta.

D. Presupuesto del proyecto

Aunque la aplicación de este proyecto requiere la utilización de diversos dispositivos mecánicos, tales como los ya mencionados en la sección 3.1.1, como su estudio no está incluido dentro del alcance de este trabajo, sus costes no han sido incluidos en el análisis presupuesto.

D.1. Costes humanos

El tiempo estimado requerido para el desarrollo de este proyecto son 300h (correspondientes a 12 créditos ECTS). El trabajo será realizado por una única persona con estudios en el ámbito de la ingeniería. Por lo tanto el precio estimado de su trabajo será:

Recurso	Número de recursos	Precio/Hora de trabajo [€/h]	Horas [h]	Total [€]
Personas	1	15	300	4.500

Tabla 10: Costes humanos del proyecto

D.2. Otros Recursos

Para desarrollar el diseño del modelo será necesario:

Recurso	Número de recursos	Precio/Unidad [€/u]	Total [€]
Ordenador HP 15s-fq1033ns	1	700	700
Licencia de Matlab	1	800	800
Toolboox	1	20	20
		TOTAL [€]	1.520

Tabla 11: Coste de los recursos utilizado

D.3. Total

Finalmente la suma total de todos los costes relacionados con el proyecto son:

Recursos humanos [€]	Otros recursos [€]	Coste total [€]
4.500	1.520	6.020

Tabla 12: Coste total del proyecto